Оглавление

ПРАКТИКУМ

Предисловие

По данным международной организации Internet World Stats (IWS) количество пользователей Интернета в 2019 году во всем мире превысило 4 миллиарда человек и составило более 53% всего населения Земли, а в 2000 году этот показатель составлял лишь 200 миллионов. Такой бурный рост числа пользователей вызван постоянно растущим количеством и качеством сервисов, которые можно получать через Интернет. Среди этих сервисов и традиционная электронная почта, и популярнейшие социальные сети, и различные коммуникационные средства — мессенджеры, системы IP-телефонии и т. п.

Однако по-прежнему среди всех Интернет-сервисов особое место отводится возможностям быстрого поиска и удобного предоставления пользователям информации, размещенной во всемирной паутине, или World Wide Web (WWW). Технологии сетевого взаимодействия, которые традиционно использовались для работы поисковых систем в среде WWW, постепенно распространяются и начинают применяться в реализации самых разных сетевых приложений и сервисов.

Вместо отдельных почтовых сервисов используются почтовые средства, интегрированные в поисковые системы, традиционные офисные приложения заменяются браузерными, которые обеспечивают такую же полную функциональность.

Корпоративные сайты, порталы и web-приложения большинства предприятий и организаций являются уже не просто Интернет-витринами. Они становятся необходимым условием конкурентоспособности и развития компаний, предоставляют возможность организации эффективных систем электронной коммерции, вертикальной и горизонтальной интеграции. Важно также, что для внутреннего корпоративного управления в настоящее время также используются системы, основанные на принципах и технологиях web-программирования. Это зачастую более эффективно, чем использование традиционных клиент-серверных систем, так как лучше отвечает условиям,

существующим в современной расширяемой и динамичной бизнес-среде.

Каждый год мы наблюдаем, как улучшаются и развиваются Интернет-сервисы, которые государство предоставляет своим гражданам, предприятиям и организациям.

Все вышеперечисленное становится возможным благодаря очень быстрому развитию технологий разработки web-приложений. В их основе лежат известные принципы организации сети Интернет и протоколы взаимодействия внутри этой сети. Однако за годы развития этих технологий возникло множество языков, сред программирования, методов и технологий, которые позволяют создавать более привлекательные, динамичные, быстрые и эффективные приложения. Описание принципов, возможностей и основ применения таких технологий и является целью данного учебного пособия.

Чтобы использовать весь накопленный потенциал, современный разработчик web-приложений должен иметь представление о принципах организации Интернета, системе адресации и базовых протоколах передачи данных, принципах разработки отдельных web-страниц и сайтов в целом, базовом языке разметки web-страниц HTML, технологиях описания стилей этих страниц (CSS). При этом современный web-сайт уже не может быть статичным и просто отображать некоторую информацию. Пользователь хочет иметь возможность обмениваться информацией, настраивать сайт под свои нужды, развивать его. Поэтому в данном пособии предусмотрено изучение технологий клиентского и серверного программирования.

Кроме того, практическая разработка web-сайтов сегодня невозможна без использования готовых библиотек и фреймворков, которые значительно облегчают процесс программирования и повышают качество создаваемого продукта. Основы применения таких инструментов также будут рассмотрены в ходе изложения.

И наконец, в современной web-разработке широко применяются системы управления содержанием сайтов (CMS-системы), позволяющие «собирать» сайт из некоторого набора готовых шаблонов. Рассмотрение этих технологий также будет предметом нашего внимания.

Теоретический материал, представленный в данном учебном пособии, подкрепляется практическими заданиями, предложенными во второй части издания — практикуме.

В результате изучения изложенного в пособии материала студенты должны:

знать

- принципы организации современных Интернет-технологий,
- основные протоколы передачи информации по сети Интернет,
- системы адресации в сети Интернет,
- основы организации клиент-серверного взаимодействия,
- основы организации системы WWW,
- основные языковые, программные и инструментальные средства создания web-приложений;

уметь

- применять современных языковые средства для разметки и оформления web-страниц,
- создавать статические и динамические web-сайты,
- создавать сайты, адаптируемые к отображению на разных типах устройств,
- использовать клиентские и серверные технологии сбора, хранения, обработки и передачи информации при организации web-приложений;

владеть

- навыками, связанными с выбором и применением наиболее эффективных инструментов разработки,
- навыками отладки и тестирования работы отдельных компонентов программ и приложений в целом.

Представленное пособие может использоваться для изучения теоретической и практической части дисциплин, связанных с технологиями современной web-разработки студентами специальностей Программная инженерия, Компьютерные системы и сети, Системный анализ и т. п.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Тема 1
ОСНОВНЫЕ ПОНЯТИЯ И ПРИНЦИПЫ WEB-ТЕХНОЛОГИЙ

После изучения данного раздела студент должен
знать
- принципы организации сети Интернет,
- базовые протоколы обмена информацией в глобальной компьютерной сети,
- принципы адресации,
- назначение и принципы организации системы доменных имен,
- сущность и принципы организации сети WWW,
- назначение и принцип работы протокола HTTP,
- проблемы и основные направления организации систем обеспечения безопасности в сети Интернет,
- сущность cookies;

уметь
- использовать адреса ресурсов Интернет,
- распознавать содержимое HTTP-запросов,
- распознавать базовые и прикладные протоколы передачи данных,
- соотносить прикладные протоколы с использующими их типами прикладных приложений.

1.1. Интернет как среда для web-взаимодействия

Всемирная паутина, или система гипертекстовых документов **World Wide Web (WWW)** является одним из важнейших сервисов, благодаря которому пользователи могут быстро находить и использовать информацию, хранящуюся на множестве различных компьютеров в любой части Интернета. Поэтому для понимания основных принципов организации и функционирования WWW необходимо, прежде всего, понять принципы организации и функционирования Интернета.

Итак, **Интернет** — это глобальная всемирная компьютерная сеть, не имеющая единого центра управления, но работающая по единым правилам и предоставляющая своим пользователям единый, постоянно расширяюшийся набор услуг.

Структура Интернета может быть представлена как совокупность связанных между собой более мелких компьютерных сетей. При этом в сети существуют организации, называемые **поставщиками услуг Интернета** (ISP, *Internet Service Provider*). Они владеют **Интернет-серверами** и **каналами передачи данных** разных уровней. ***Глобальные провайдеры*** предоставляют свои ресурсы ***региональным провайдерам***, те, в свою очередь, ***локальным провайдерам***, которые уже предоставляют Интернет-ресурсы и сервисы **конечным пользователям** (частным лицам или организациям).

Проблема организации взаимодействия состоит в том, что компьютеры, подключенные к Интернет, различаются аппаратной или программной системой (платформой), имеют разное прикладное программное обеспечение, разные средства для физического соединения с сетью и т. д. Таким образом, Интернет является очень сложной структурой, и соответственно, такой же сложной является задача организации стабильного взаимодействия между ее узлами.

Для решения этой задачи был использован многоуровневый подход, который рассматривает процесс взаимодействия между двумя любыми компьютерами, подключенными к сети Интернет, как последовательное применение некоторого единого пакета технологий. Каждый уровень этого процесса позволяет выполнить одну из подзадач процесса взаимодействия, а затем передать результат выполнения этой подзадачи на следующий этап. Основная идея взаимодействия состоит в том, что каждый уровень должен обеспечивать интерфейс не только с соседними уровнями, но и с аналогичным уровнем второго узла.

Такой пакет технологий, обеспечивающий «взаимопонимание» любых двух компьютеров, подключенных к сети Интернет называется ***стеком Интернет-протоколов.***

Здесь уместно привести известную аналогию — дипломатический протокол. Взаимодействие между представителями государств на международных встречах осуществляется строго на соответствующем уровне: президент — президент, премьер-министр —

премьер-министр и т. п. При этом внутри делегации происходит обмен информацией между различными уровнями иерархии.

Таким образом, сложная задача взаимодействия различных узлов в сети разбивается на ряд более простых задач.

1.2. Основные Интернет-протоколы

Рассмотренный нами выше принцип декомпозиции задачи взаимодействия между любыми двумя компьютерами в сети Интернет является основой существующей модели сетевого взаимодействия, которая получила название **TCP/IP** (происходит от названий двух важнейших протоколов семейства — *Transmission Control Protocol* (TCP) и *Internet Protocol* (IP)).

Эта модель предполагает, что взаимодействие реализуется через четыре уровня:

— самый верхний из них, прикладной, обеспечивает обработку информации, сформированной отправителем и поступающей от какой-либо прикладной программы, или информации, которая должна быть принята соответствующей прикладной программой и продемонстрирована получателю;

— ниже располагается транспортный уровень, обрабатывающий связь между узлами;

— еще ниже располагается сетевой (межсетевой) уровень, обеспечивающий межсетевое взаимодействие между независимыми сетями;

— и на самом нижнем уровне, канальном, определяются методы связи для данных, которые остаются в пределах одного сегмента сети.

Для пояснения принципа работы стека протоколов TCP/IP чаще всего используется аналогия с работой обычной почтовой системы. Письмо перед отправкой снабжается почтовыми адресами получателя и отправителя. Адрес получателя анализируется в ближайшем почтовом отделении, после чего письмо отправляется далее через цепочку других почтовых отделений до тех пор, пока не достигнет адресата. В конечном почтовом отделении контролируется сохранность и конфиденциальность почтового отправления.

Каждый компьютер, подключенный к сети Интернет тоже имеет уникальный адрес, который называется **IP-адресом**

(*Internet Protocol Address*). IP-адрес состоит из четырех десятичных чисел от 0 до 255, разделенных точкой (например 195.34.32.116.). Кроме того, в зависимости от того, какая именно программа (например, браузер, Skype или почтовый клиент) создала передаваемое сообщение, адрес дополняется также ***номером порта*** (каждая программа традиционно привязана к определенным портам). Сохранение номера порта позволяет адресату узнать, какая программа сможет «принять» передаваемое сообщение. Комбинация IP-адреса и номера порта называется ***сокетом.***

Далее сообщение, передаваемое через Интернет, проходит несколько уровней, при этом задействуются различные составляющие стека Интернет-протоколов.

На верхнем, **прикладном, уровне** работают такие протоколы, как **HTTP** (для передачи как раз гипертекста в среде WWW), **FTP** (для передачи файлов), **SMTP** и **POP3** (для приема и передачи электронных писем, созданных при помощи программ-почтовых клиентов) и т. п. Подробнее мы рассмотрим некоторые из них ниже, однако необходимо указать, что эти протоколы, для своего типа передаваемой информации, позволяют разделять сообщение на пакеты определенной длины.

TCP (*Transmission Control Protocol* — протокол управления передачей) и **UDP** (*User Datagram Protocol*, протокол датаграм пользователя) — это протоколы транспортного уровня. На этом уровне, полученные от прикладного уровня пакеты снабжаются IP-адресом получателя и некоторой другой вспомогательной информацией, к пакету добавляется порт отправителя и порт получателя.

TCP — это протокол с установлением соединения и с гарантированной доставкой пакетов. Сначала производится обмен специальными пакетами для установления соединения, далее по этому соединению посылаются пакеты, причем с проверкой, дошел ли пакет до получателя. Если пакет не дошел, то он посылается повторно.

UDP — это протокол без установления соединения и с негарантированной доставкой пакетов.

Протокол **IP** (*Internet Protocol*, межсетевой протокол) — это протокол сетевого уровня. Именно он отвечает за прохождение каждого пакета по цепочке свободных в данный момент участков сети от компьютера отправителя к компьютеру получателя. Номера портов на сетевом уровне не используются. Какому

порту, т. е. приложению адресован этот пакет, был ли этот пакет доставлен или был потерян, на этом уровне неизвестно — это не его задача, это задача транспортного уровня.

Канальный уровень — уровень передачи данных внутри одного сегмента сети. В стеке протоколов TCP/IP он специально не определен, но поддерживает все современные стандарты организации сетей на физическом уровне: Ethernet, Token Ring, FDDI, Fast Ethernet. На этом уровне выполняется:

— перевод IP-адресов в физические адреса компьютеров;

— определение метода доступа к среде передачи, то есть способа, с помощью которого компьютер устанавливает свое право на передачу данных;

— обработка возможных коллизий передачи;

— определение того, в какой именно физической среде происходит передача данных (воздух, оптоволокно, коаксиальный кабель и т. п.).

После доставки пакета получателю он обрабатывается всем стеком протоколов, но в обратном порядке — принимается транспортными протоколами и, в случае успешного контроля качества передачи, передается на прикладной уровень и предоставляется пользователю при помощи соответствующего приложения.

1.3. Система доменных имен DNS

Важнейшую роль при организации взаимодействия в сети Интернет имеет система именования узлов. Каждый компьютер, подключенный к Интернету, должен иметь уникальное имя, но в некоторых случаях это имя должно быть запоминающимся и отражающим некоторый смысл.

Как мы все знаем из школьного курса информатики, компьютер, как техническое устройство, способен хранить информацию в двоичной цифровой форме. Базовая адресация каждого Интернет-узла, IP-адрес — это цифровое имя, состоящее из четырех чисел от 0 до 255. Однако пользователи — не машины, и для некоторых узлов сети, к которым часто обращается множество пользователей, наряду с цифровыми, выделяются буквенные имена, которые называются также ***доменными именами***.

Для установления соответствия между доменным именем и IP-адресом используется специальная система доменных имен (**DNS**, *Domain Name System*), которая основана на специ-

альных таблицах соответствия, которые хранятся на специальных ***DNS-серверах*** в сети Интернет.

В сетях TCP/IP используется доменная система имен, имеющая иерархическую (древовидную) структуру. Данная структура имен напоминает иерархию имен, используемую во многих файловых системах. Запись доменного имени начинается с самой младшей составляющей, затем после точки следует следующая по старшинству символьная часть имени и так далее. Последовательность заканчивается корневым именем, например, company.yandex.ru.

Построенная таким образом система имен позволяет разделять административную ответственность по поддержке уникальности имен в пределах своего уровня иерархии между различными организациями.

Совокупность имен, у которых несколько старших составных частей совпадают, образуют ***домен имен***.

Корневой домен управляется центральными органами Интернета: IANA и Internic.

Домены верхнего уровня назначаются для каждой страны, а также для различных типов организаций. Для обозначения стран используются двухбуквенные аббревиатуры, например, ru (Российская Федерация), us (США), it (Италия), fr (Франция).

Для различных типов организаций используются трехбуквенные аббревиатуры:

— net — сетевые организации;
— org — некоммерческие организации;
— com — коммерческие организации;
— edu — образовательные организации;
— gov — правительственные организации.

Перечисленные типы доменов верхнего уровня являются наиболее распространенными. Однако есть и другие варианты, которые вы можете встретить.

Для получения доменного имени необходимо зарегистрироваться в соответствующей организации, которой организация InterNIC делегировала свои полномочия по распределению доменных имен.

1.4. Структура и принципы организации WWW

Большинство ресурсов WWW основано на технологии ***гипертекста.*** Эту технологию предложил Тим Бернерс-Ли, ко-

торый в 1989 г. реализовал идею связывания публикуемых документов ***гиперссылками*** на другие документы, что позволило значительно ускорить процессы поиска информации. Гипертекстовые документы, размещаемые во Всемирной паутине, называются ***web-страницами.*** Несколько web-страниц, объединенных общей темой и дизайном, а, также связанных между собой гиперссылками и обычно находящихся на одном и том же web-сервере, называются ***web-сайтом.***

Для задания ссылок на отдельные ресурсы (как правило, отдельные файлы) в WWW используются идентификаторы ресурсов **URI** (*Uniform Resource Identifier*). Для определения местонахождения ресурсов в сети используются локаторы ресурсов **URL** (*Uniform Resource Locator*). Такие URL-локаторы сопоставляют URI с их IP-адресами в системе доменных имен DNS. Основы использования универсальных идентификаторов и локаторов ресурсов были также заложены в работах Т. Бернса-Ли, Р. Кайо и нек. др.

1.5. Протокол HTTP

Как упоминалось выше, HTTP — это протокол прикладного уровня для передачи гипертекста.

Протокол HTTP использует web-сервер — программу для хранения web-сайтов и обработки запросов пользователя на предоставление ему определенного web-ресурса, и программу-клиент, которая способна формировать запросы к серверу и отображать результаты этих запросов.

Центральным объектом в HTTP является ресурс, на который указывает URL в запросе клиента. Обычно такими ресурсами являются хранящиеся на web-сервере файлы.

Наиболее популярными реализациями web-серверов являются: Internet Information Services (IIS), Apache, ligHTTPd, nginx. Клиентами служат браузеры, например, Google Chrome, Opera, Mozilla Firefox, Safari и другие.

Классическая схема HTTP-сеанса предполагает установление TCP-соединения, запрос клиента, ответ сервера, разрыв TCP-соединения. Обычно запрос клиента представляет собой требование передать HTML-документ или какой-нибудь другой ресурс, а ответ сервера содержит код этого ресурса.

На рис. 1.1 представлен типичный HTTP-запрос.

```
1.  GET http://www.MyOrganization.com/ HTTP/1.1
2.  Host: www. MyOrganization.com
3.  Connection: keep-alive
4.  Accept: text/html,application/xhtml+xml,application/xml;
    q = 0.9,*/*;q = 0.8
5.  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
6.  Accept-Encoding: gzip,deflate,sdch
7.  Accept-Language: en-US;q = 0.8,en;q = 0.6
8.  Cookie: __gads = ID = 42213729da4df8df:T = 1368250765:S =
    ALNI_MaOAFe3U1T9Syh;
9.  (пустая строка)
10. (тело запроса)
```

Рис. 1.1. **Структура HTTP-запроса**

Начальная (первая) строка указывает метод запроса (GET или POST), строку URL и версию протокола HTTP. Необязательные заголовки (строки 2—8) характеризуют сообщение, параметры передачи и предоставляют другую мета-информацию. В строке 10 представлено необязательное тело сообщения, содержащее его данные, оно отделено от заголовков пустой строкой.

Метод *GET* служит для получения любой информации, идентифицированной URI-запроса. Другим распространенным методом в HTTP-запросах является метод *POST*, который позволяет отправлять данные для обработки на указанный ресурс.

Теперь рассмотрим ответ на представленный запрос (рис. 1.2).

```
1.  HTTP/1.1 200 OK
2.  Cache-Control: private
3.  Content-Type: text/html
4.  Content-Encoding: gzip
5.  Vary: Accept-Encoding
6.  Server: Microsoft-IIS/7.5
7.  Set-Cookie: ASPSESSIONIDQQRBACTR = FOCCINICEFAMEKODNKIBFOJP;
    path = /
8.  X-Powered-By: ASP.NET
9.  Date: Sun, 04 Aug 2013 13:33:59 GMT
10. Content-Length: 8434
11. (пустая строка)
12. (содержимое страницы)
```

Рис. 1.2. **Структура HTTP-ответа**

Первая строка представляет собой версию протокола HTTP и код статуса ответа. Код 200 означает, что ресурс был най-

ден, а в случае неудачного запроса будет возвращен код 404, 403 или какой-либо другой. Необязательные заголовки (строки 2—10) предоставляют различную мета-информацию об ответе. В строке 12 представлено тело сообщения, следующее за заголовками, которое должно быть отделено от них пустой строкой. Тело ответа, как правило, содержит HTML-код запрашиваемой веб-страницы.

В отличие от многих других протоколов, HTTP является протоколом без памяти. Это значит, что в процессе взаимодействия не сохраняется информация о предыдущих запросах клиентов и ответах сервера. Однако все пользователи Интернета замечают, что после ввода и обработки некоторого запроса браузер запоминает эту информацию, и впоследствии предоставляет ее в форме рекламы, помощи в строках ввода и т. п. Это становится возможным благодаря использованию механизма ***cookie,*** который позволяет серверу хранить информацию на компьютере клиента и извлекать ее оттуда.

Инициатором записи cookie выступает сервер. Если в ответе сервера присутствует поле заголовка Set-cookie, клиент-браузер воспринимает это как команду на запись cookie. В дальнейшем, если клиент обращается к серверу, от которого он ранее принял поле заголовка Set-cookie, помимо прочей информации он передает серверу данные cookie.

1.6. Безопасность HTTP

Поскольку протокол HTTP предназначен для передачи символьных данных в открытом (незашифрованном) виде, то лица, имеющие доступ к каналу передачи данных между клиентом и сервером, могут без труда просматривать весь трафик и использовать его для совершения несанкционированных действий. В связи с этим был разработан ряд расширений базового протокола, направленных на повышение защищенности интернет-трафика от несанкционированного доступа.

Наиболее распространенным является расширение HTTPS, при котором данные, передаваемые по протоколу HTTP, «упаковываются» в криптографический протокол SSL или TLS, тем самым обеспечивая защиту этих данных. Чтобы подготовить web-сервер для обработки HTTPS соединений, администратор должен получить и установить в систему сертификат для этого web-сервера.

Контрольные вопросы

1. Как вы можете определить понятие WWW?
2. Почему важны стандарты Интернета?
3. Что такое стек Интернет-протоколов?
4. Какова роль протоколов TCP/IP в информационном обмене в сети Интернет?
5. Какие протоколы прикладного уровня вам известны?
6. Как организована система адресации в Интернете? Для чего нужна система доменных имен?
7. Какой основной протокол прикладного уровня используется в системе WWW?
8. Какова структура запроса HTTP?
9. Как обеспечивается безопасность передачи данных в Web?
10. Что такое cookie?

Тема 2
ОСНОВЫ ЯЗЫКА РАЗМЕТКИ ГИПЕРТЕКСТА HTML

После изучения данного раздела студент должен:

знать

- назначение и основные особенности языка HTML,
- правила описания структуры документа HTML,
- основные теги описания элементов страницы на языке HTML,
- принципы описания свойств элементов,
- дополнения к языку, введенные в версии HTML5;

уметь

- описывать служебную информацию о документе в области заголовка HTML-документа,
- описывать содержимое web-страницы с использованием тегов языка HTML,
- структурировать документ с применением тегов заголовков, абзацев, таблиц, списков, блоков,
- иллюстрировать страницы с использованием изображений,
- организовывать гиперссылки на другие HTML-документы,
- создавать формы пользователя;

владеть

- навыками выделения структурных блоков HTML-страницы с примененением таблиц и блоков.

2.1. Назначение и особенности HTML

Кроме платформенно-независимых средств передачи информации в системе WWW, включающих использование стека протоколов TCP/IP и прикладного протокола передачи гипертекста HTTP, для успешного развития этого сервиса требовался удобный, понимаемый всеми браузерами, развивающийся язык, на котором можно было бы создавать гипертекстовые

документы. Поэтому одним из главных компонентов технологии создания распределенной гипертекстовой системы World Wide Web стал язык гипертекстовой разметки **HTML** (*HyperText Markup Language*). Удобство его применения обусловлено следующими двумя его *особенностями*.

1. Формат такого гипертекстового документа — обычный *текстовый файл*. Поэтому его можно создавать с помощью любого текстового редактора (в простейшем случае — блокнота), хотя в настоящее время создано множество специализированных редакторов, позволяющих быстрее и с меньшим количеством ошибок создавать и структурировать такие документы.

2. В HTML реализована ***теговая модель*** описания документа. То есть структура и содержимое документа определяются специальными метками — тегами, внутри которых помещают информацию о том, как браузер должен трактовать данный блок.

Таким образом, ***сайт в концепции WWW*** — это набор текстовых файлов, размеченных на языке HTML, который определяет форму представления информации (разметка) и структуру связей между этими файлами и другими информационными ресурсами (гипертекстовые ссылки).

Из курсов информатики и программирования мы знаем, что любой интерпретируемый язык высокого уровня требует перевода его инструкций на язык команд, которые могут быть понятны компьютерной системе. В World Wide Web функции интерпретатора разделены между web-сервером и интерфейсом пользователя, функции которого реализованы в браузере. Сервер, кроме доступа к документам и обработки гипертекстовых ссылок, обеспечивает предпроцессорную обработку документов, в то время как браузер осуществляет интерпретацию конструкций языка, связанных с представлением информации.

Итак, документ, написанный (иногда говорят «размеченный») с использованием языка HTML — это просто текст, разделенный на отдельные структурные или содержательные *элементы*. Эти элементы окружены одинаковыми открывающими и закрывающими командами — тегами. Каждый тег начинается и заканчивается с угловых скобок (< и >). Структуру любого тега можно рассмотреть на следующем примере:

```
<p>Hello world!</p>
```

Здесь используется тег <p>, который позволяет выделить отдельный абзац, содержащий тот текст, который находится между «открывающей» частью тега <p> и ее «закрывающей» частью </p>.

Многие теги можно снабжать атрибутами, которые дополняют сведения о том, как именно браузер должен отобразить данный элемент.

В качестве ***общих правил написания HTML документов*** необходимо отметить следующие:

— нужно выделять новую строку для каждого блочного, табличного или списочного элемента;

— ставить отступы для каждого дочернего элемента, отступы должны составлять минимум 2 пробела;

— не следует использовать заглавные буквы для тегов, только строчные;

— всегда нужно использовать закрывающий тег;

— в начале документа желательно указывать его тип:

```
<! DOCTYPE html>
```

— для комментирования текста программы следует использовать символьные скобки <!   >.

Как было сказано выше, HTML активно развивается. На момент написания данного учебного пособия актуальной версией является HTML5. Поэтому в дальнейшем изложении мы рассмотрим базовые возможности языка HTML, а, при необходимости будем указывать на особенности, связанные с переходом на версию HTML5.

Далее рассмотрим структуру и назначение тегов языка HTML более подробно.

2.2. Структура документа на HTML

Итак, каждый отдельный документ HTML может быть рассмотрен как контейнер, внутри которого размещается все его содержимое. Все содержимое файла помещается между открывающим и закрывающим тегом <html>.

```
<! DOCTYPE html>
  Содержание документа
</html>
```

Контейнер html или гипертекстовый документ содержит два других вложенных контейнера: заголовок документа (***head***) и тело документа (***body***). Рассмотрим простейший пример классического документа.

```
<!DOCTYPE html>
  <head>
    <title>Мой первый документ</title>
  </head>
  <body>
    <h1> Заголовок документа </h1>
    <p>Это содержимое единственного абзаца документа</p>
  </body>
</html>
```

Заголовок HTML-документа (то, что помещено внутрь контейнера head) является необязательным элементом разметки. Но, для быстрой индексации страницы поисковыми машинами, продвижения этого ресурса в списках выдачи поисковых систем, корректности отображения документов браузерами служебная информация, содержащаяся в области заголовка имеет очень важное значение.

Внутри тегов head могут размещаться следующие элементы, окруженные в свою очередь, своими тегами.

Элемент разметки ***title*** служит для именования вкладки окна браузера, в котором просматривается документ. Роботы многих поисковых систем используют содержание элемента title для создания поискового образа документа. Слова из title попадают в индекс поисковой системы. Поэтому элемент *title* всегда рекомендуется использовать на страницах web-сайта.

Элемент разметки ***base*** служит для определения базового URL для гипертекстовых ссылок документа, заданных в неполной (частичной) форме.

```
<base href = //cfu.crimea-ru/start/>
<base href = ../next_level/document.HTML>
```

Во втором случае в качестве базы по умолчанию выбирается каталог, в котором размещен HTML-документ (../).

Элемент разметки ***meta*** содержит управляющую информацию, которую браузер использует для правильного отображения и обработки содержания тела документа.

Для определения кодировки документа:

```
<meta charset = "utf-8">
```

Для указания информации о документе:

```
<meta name = "author" content = "Системный анализ">
```

Для описания страницы и задания ключевых слов:

```
<meta name = "description" content = "Методы системного анализа">
<meta name = "keywords" content = "моделирование, системный
анализ, системный подход, синтез, принятие решений">
```

Для указаний роботу[1] индексировать, отслеживать гиперссылки на странице:

```
<meta name = "robots" content = "index, follow">
```

Элемент разметки ***link*** наиболее часто используется для подгрузки файлов, содержащих стили оформления документа.

```
<link rel = stylesheet href = "../css/style.css" type =
"text/css">
```

В данном случае речь идет о загрузке стилей из файла style.css, который хранится в виде каскадной таблицы стилей (stylesheet) по адресу, указанному в динамической ссылке при помощи атрибута href (подробнее об этом мы поговорим в следующем разделе).

Элемент разметки ***style*** подключает внутренние таблицы стилей.

```
<style type = "text/css">
  p{
    background-color: #ef4444;
    color: #666666;
  }
</style>
```

[1] Поисковый робот — специальная программа, принадлежащая определенной поисковой системе и предназначенная для индексирования (занесения в базу сайтов поисковой системы) веб-сайтов и их страниц, найденных в интернет-пространстве.

В этом примере указано, что для всех абзацев (тег <p>) должны быть применены заданный цвет фона и шрифта.

После завершения описания заголовка HTML-документа и закрытия элемента разметки </head>, открывается элемент разметки ***body*** (тело документа), внутри которого располагаются все элементы, которые и составляют описываемую страницу и видны пользователю при просмотре ее в браузере.

2.3. Основные теги тела документа HTML

Базовые теги, позволяющие описать структуру и содержание тела HTML-документа, а, также дополнительные теги, появившиеся в версии HTML5, представлены в табл. 2.1.

Открывающие теги могут содержать ***атрибуты.*** С их помощью можно менять свойства объекта, которому соответствует тег. Порядок следования атрибутов в теге не важен. Атрибуты записываются внутри открывающего тега через пробел от его имени в виде отдельного ключевого слова или ключевого слова, знака « = » и параметра (значения атрибута). Атрибут меняет свойство элемента, работающее по умолчанию. Он действует от открывающего тега, в котором он задан, до закрывающего или только внутри тега, если тег не имеет парного.

В версии HTML5 появилось несколько новых тегов, которые помогают автоматически распознавать структуру документа. Вместо <div class = «header»> стоит писать просто <header>. Аналогичным образом можно использовать теги <article>, <aside>, <footer>, <nav>, которые заменяют привычный <div>. Такие теги позволяют более точно описать содержание основных функциональных разделов страницы.

Ниже приведен пример простой страницы, размеченной с использованием блоков и встроенных стилей.

```
<!-- Начало Web-страницы -->
<!DOCTYPE html>
<html>
  <!-- Начало заголовка Web-страницы -->
  <head>
    <!-- Описание метаданных, т. е. информации о Web-странице. -->
    <META HTTP-EQUIV = "Content-Type" CONTENT = "text/html;
charset = UTF8">
    <!-- Описание надписи на вкладке браузера -->
    <title>Пример</title>
```

Таблица 2.1

Базовые теги HTML

Теги	Назначение	Пример	Описание результата
- 	6 уровней заголовков от h1 до h6	заголовок 1 заголовок 2	заголовок 1 заголовок 2
	Разделение текста на абзацы	 Это первый абзац Это второй абзац 	Это первый абзац Это второй абзац
<a>	Вывод гиперссылок	 Перейти на второй документ 	Перейти на второй документ (в виде ссылки, которая ведет на документ с именем 2.html)
<img>	Вывод изображений	<img src = "img/image.gif" alt = "изображение">	Будет выведено изображение, находящееся в файле с именем image.gif. При отсутствии изображения появится соответствующий альтернативный текст
	Начало и конец нумерованного списка	 - первый пункт - второй пункт - первый пункт - второй пункт 	1) первый пункт 2) второй пункт — первый пункт — второй пункт
	Начало и конец маркированного списка		
	Начало и конец пункта списка любого типа		

Окончание табл. 2.1

Теги	Назначение	Пример	Описание результата
`<table></ table>`	Начало и конец таблицы	`<table border width = 20%>` `<tr>` `<td >` `1 столбец 1 строки` `</td>` `<td >` `2 столбец 1 строки` `</td>` `</tr>` `<tr>` `<td colspan = 2>` `Colspan используем` `если нужно` `объединять столбцы` `</td>` `</tr></table>`	1 Столбец 1 Строки | 2 Столбец 1 Строки Colspan используем если нужно объединять столбцы
`<tr></tr>`	Начало и конец строки таблицы		
`<td></td>`	Начало и конец столбца таблицы в пределах строки		
`<div>` `</div>`	Разделение страницы на блоки	`<div id = "header">` `Header` `</div>`	Область заголовка описывается блоком с идентификатором header, внутри которого может быть любое наполнение

```
<!-- Описание стилей документа -- >
    <style type = "text/css">
      h1{
        /* Описание цвета текста заголовка 1 уровня*/[1]
        color: red;
        /* Размер шрифта */
        font-size: 26px;
        /* Тень текста */
        text-shadow: 2px 2px 5px white;
        /* Тень блока */
        box-shadow: 2px 2px 5px blue;
        /* Выравнивание текста */
        text-align: center;
        /* Цвет фона */
        background: gray;
        /* Ширина объекта */
        width: 560px;
      }
      body{
        /* Описание цвета текста */
        color: blue;
      }
      p{
        /* Размер шрифта для обычных абзацев*/
        font-size: 16px;
      }
    </style>
  </head>
  <!-- Описание объектов Web-страницы -->
  <body>
    <!-- Заголовок 1 типа -->
    <h1>Это пример заголовка</h1>
    <!-- Обычный текст -->
    <p>Это пример текста страницы</p>
  </body>
</html>
```

В результате получим следующую страницу:

Рис. 2.1. **Пример простой страницы с использованием встроенной таблицы стилей**

[1] При разметке документа следует учитывать, что формат комментариев в блоке описания стилей отличается от используемого в остальной части документа. — *Прим. ред.*

2.4. Формы HTML

Важнейшей особенностью языка разметки web-страниц является возможность организации пользовательского интерфейса. Современный сайт должен не только предоставлять пользователю актуальную, важную и удобно читаемую информацию, но и получать от него ответную реакцию. Для этого служит группа тегов, которые обычно объединяются внутри тегов <form> и </form>

Тег **<form>** выделяет фрагмент документа как форму и определяет границы использования других тегов, размещаемых в форме. Атрибуты тега <form> определяют, какой файл или программа будут обрабатывать полученную из формы информацию, а также, каким методом будет осуществлен HTTP-запрос. Обычно это методы GET или POST (см. первый раздел).

Например:

```
<form method = POST action = "result.php">
```

Здесь предполагается, что данные формы будут переданы для обработки файлу с именем result.php методом POST.

Основной тег, который может использоваться внутри формы — это тег <input></input>.

Тег **<input>** используют для определения области внутри *формы*, куда вводятся данные. Это может быть текстовое поле, опция, изображение или кнопка. Вид поля ввода определяется значением атрибута ***type***. Рассмотрим основные из этих типов.

Атрибут type = "text". Когда пользователю необходимо ввести небольшое количество текста (одну или несколько строк), используется тег <INPUT>, и атрибут type устанавливается в значение «text». Это значение принято по умолчанию и указывать его необязательно. Кроме того, задается атрибут ***name*** для определения имени этого поля ввода, для дальнейшей обработки полученного из него значения. Имеется еще три дополнительных атрибута, которые можно использовать. Первый называется ***maxlength***, он ограничивает число символов, вводимых пользователем в текущее поле. По умолчанию данное число не ограничено. Вторым атрибутом является ***size***, определяющий размер видимой на экране области, занимаемой полем. Значение по умолчанию определяется типом браузера. Если значение ***maxlength*** больше, чем ***size***, браузер будет про-

кручивать данные в этом окне. Последним из дополнительных атрибутов является атрибут ***value***, обеспечивающий значение поля ввода по умолчанию.

```
<p>
  ваше имя: <input name = "FIO" size = 35 value =
  "Фамилия И. О.">
</p>
```

Результат:

ваше имя: Фамилия И.О.

Атрибут type = "***checkbox***" используется для создания независимых флажков. При этом указываются также атрибуты name и value. В некоторых случаях необходимо инициализировать данный флаг как уже отмеченный. В таких случаях тег <input> должен содержать атрибут ***checked***:

```
<p>
  Язык 1: <input name = "язык1" type = "checkbox" value = "HTML">
</p>
<p>
  Язык 2: <input name = "язык2" type = "checkbox" value =
  "JScript" checked >
</p>
```

Результат:

Язык 1: ☐

Язык 2: ☑

Атрибут type = "radio" применяют, когда требуется организовать выбор одного из нескольких возможных значений. Для этого в теге <input> должны быть указаны атрибуты name и value. Но атрибут name должен содержать одно и тоже значение для всех вариантов выбора.

```
<p>
  Пол мужской: <input name = "пол" type = "radio"
  value = "мужской">
</p>
```

```
<p>
   Пол женский: <input name = "пол" type = "radio" value = "жен-
ский">
</p>
```

Результат:

Пол мужской:

Пол женский:

Атрибут type **= "*password*".** Применяют для ввода пароля. Используя данный тип, можно организовать ввод пароля без вывода на экран составляющих его символов. При этом следует помнить, что введенные данные передаются по незащищенным каналам связи и могут быть перехвачены.

```
<p>
  введите логин:<input name = "login">
</p>
<p>
  Введите пароль:<input type = "password" name = "pass">
</p>
```

Результат:

введите логин: Student

Введите пароль: ••••

Атрибут type **= "*reset*".** Когда пользователь заполняет форму, ему может потребоваться начать все сначала. Для этого существует кнопка Reset, щелкнув по которой, пользователь может вернуться к первоначальным значениям полей.

```
<input type = "reset" value = "очистить форму">
```

Результат:

очистить форму

Атрибут type = "***submit***" используется для организации подтверждения отправки формы. Браузер выводит данный элемент как кнопку, по которой пользователь может щелкнуть, чтобы завершить процесс редактирования. Важно в этом случае использовать атрибут name, так как с событием нажатия на этот элемент часто связывают начало программной обработки данных формы.

```
<p>
  <input type = "submit" name = "button1"
  value = "отправить данные">
</p>
```

Рассмотрим еще несколько тегов, которые могут входить в блок формы.

Тег <textarea>. В некоторых случаях может потребоваться организовать ввод большого количества текста. В таких случаях используется тег <textarea> для создания текстового поля из нескольких строк. Данный тег использует три атрибута: cols, name и rows. Атрибут ***cols*** определяет количество колонок, содержащихся в текстовой области, атрибут ***name*** определяет наименование поля, атрибут ***rows*** задает количество видимых строк текстовой области.

```
<p>
  Введите сюда Ваш отзыв:
    <textarea name = "тема" cols = "38" rows = "3">
    </textarea>
</p>
```

Результат:

Введите сюда Ваш отзыв:

Тег <select>. Когда формы HTML становятся более сложными, в них часто включают списки с прокруткой и выпадающие меню. Для этого используют тег select. Для определения списка пунктов используют тег **<option>**. Тег <select> поддерживает три необязательных атрибута: multiple, name и size. Атрибут multiple позволяет выбрать более чем одно наименование

из списка, атрибут name определяет наименование объекта, атрибут size определяет число видимых пользователю пунктов списка. Если в форме установлено значение атрибута size = 1, то браузер выводит на экран список в виде выпадающего меню. В случае size > 1 браузер представляет на экране обычный список.

Тег <option> используется только внутри тега <select>. Он поддерживает два дополнительных атрибута: ***selected*** и ***value***. Атрибут selected используется для первоначального выбора значения элемента по умолчанию, атрибут value указывает на значение, возвращаемое формой после выбора пользователем данного пункта.

```
<p>выбор:
  <select name = "выбор">
    <option value = "вариант 1">вариант 1</option>
    <option value = "вариант 2">вариант 2</option>
    <option value = "вариант 3"> вариант 3</option>
    <option value = "вариант 4" selected>вариант 4</option>
  </select>
</p>
```

Результат:

выбор: вариант 4
вариант 1
вариант 2
вариант 3
вариант 4

Таким образом, комбинируя вышеприведенные теги с различными атрибутами, можно создать HTML-форму любой сложности.

Контрольные вопросы

1. Для чего предназначен язык HTML?
2. Какие принципы, лежащие в основе языка HTML, объясняют его популярность и распространенность?
3. Каковы основные соглашения и правила, используемые при создании HTML-документа?
4. Как описывается структура HTML-документа?

5. Какая информация и при помощи каких тегов может содержаться в заголовочной части HTML-документа?
6. Какие базовые теги описания тела HTML-документа вам известны?
7. Для чего и по каким правилам используются атрибуты тегов?
8. Для чего создаются HTML-формы?
9. Перечислите основные теги HTML-форм с их основными атрибутами.

Контрольные задания

1. Создайте HTML-документ с именем Первый.html, который содержит заголовок «Язык HTML»; ниже три абзаца, содержащих основные сведения о языке HTML, оформленные разными стилями, ниже — картинку, соответствующую теме страницы; ниже — три гиперссылки на страницы с именами Второй.html, Третий.html и Четвертый.html.
2. Создайте документ с именем Второй.html, содержащий список основных тегов языка HTML (список многоуровневый, выделены теги заголовка страницы и теги тела страницы). Добавьте на страницу гиперссылки на другие страницы. Добавьте на страницу таблицу стилей, которая оформляет созданные списки с использованием различных цветов и шрифтов.
3. Создайте документ с именем Третий.html и разместите на нем таблицу, поясняющую использование тегов, которые добавлены в версию HTML 5. Добавьте на страницу гиперссылки, позволяющие переходить на страницы Первый.html, Второй.html и Четвертый.html.
4. Создайте документ с именем Четвертый.html и разместите на нем форму, которая позволит пользователю ввести свое имя, выбрать свой город из выпадающего списка и свой пол из двух предложенных вариантов. Оправка данных из формы для будущей обработки выполняется кнопкой submit. Создайте в этом документе гиперссылки на остальные документы.

Тема 3
КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ

После изучения данного раздела студент должен:

знать

- назначение стилей документов HTML,
- назначение и особенности применения каскадных таблиц стилей,
- приоритетность селекторов CSS,
- способы применения стилей к отдельным тегам, документам и сайтам;

уметь

- использовать встроенные, внедренные и внешние таблицы стилей,
- применять стили к оформлению текста, отдельных элементов страницы и блока,
- применять методы наследования и комбинирования селекторов,
- применять CSS для указания формы и расположения блоков документов;

владеть

- навыками описания стилей для тегов, классов, идентификаторов и их комбинаций,
- эффективного применения селекторов CSS для создания единого стиля сайта.

3.1. Принцип разделения контента и оформления web-документа

Как было показано в предыдущем разделе, можно довольно успешно реализовывать web-страницы и web-сайты в целом, используя только возможности HTML. Однако, чем больше становится сайт, тем больше требуется времени и других сетевых ресурсов для загрузки страниц. Разработчикам следует позаботиться о том, чтобы не загружать информацию об оформлении и компоновке страниц в каждом файле HTML. Вместо этого можно (и нужно) создавать простые файлы HTML, которые включают только содержимое страниц, а информацию об оформлении и компоновке размещать один раз, в отдельном

файле, содержащем только наборы стилей. При этом ссылки на файл, в котором хранятся эти стили, помещаются в каждый HTML-документ. Это обеспечивает также легкость поддержки сайта: если потребуется изменить внешний вид сайта, обновления нужно сделать только в одном месте.

Такой подход упрощает и совместимость с различными устройствами: Если страница не содержит информации о стилях оформления, ее можно переформатировать для разных устройств с совершенно разными атрибутами (например, размером экрана), применяя просто альтернативную таблицу стилей. Разделение контента (содержимого) и оформления также позволяет определить разные стили для разных методов представления (например, просмотра на экране или печати).

Разграничение содержимого и оформления web-страницы улучшает также продвижение сайта в поисковых системах, которые используют специальные «программы-роботы», считывающие содержимое страниц на сайтах и индексирующие их. Если эта программа не сможет прочитать содержимое страницы, перегруженной стилевой информацией, или неправильно интерпретирует важные вещи, то вероятность того, что в результатах поиска страница окажется на верхних позициях, существенно уменьшится.

3.2. Основы CSS

Каскадные таблицы стилей (*Cascading Style Sheets*, CSS) — это формальный язык описания внешнего вида документа, созданного с использованием языка разметки HTML.

Мы уже встречались с описанием стилей отдельных тегов или блоков в предыдущем разделе, и при этом речь шла о ***встроенных CSS***, если они указаны прямо как атрибуты тегов, или о ***внедренных***, если стили собраны внутри блока <style>…</style> в области заголовка HTML-документа.

Для подключения ***внешней*** **(*связанной*)** таблицы стилей к документу в области заголовка страницы используется тег **<link>**:

```
<!DOCTYPE html>

  <head>
    <link href = "mystyle.css" rel = "stylesheet" type = "text/css">
  </head>
```

```
  <body>
    <p>в этом абзаце текст должен быть зеленым, окруженным красной рамкой</p>
  </body>
</html>
```

Следующая таблица стилей CSS (хранящаяся, например, в файле *mystyle.css* в той же папке, что и HTML-документ) устанавливает цвет текста всех абзацев зеленым и окружит его сплошной красной рамкой:

```
p {
   color : green;
   border: solid red;
}
```

Термин «***каскадные***» в отношении таблиц стилей указывает на то, что HTML-документ может использовать несколько стилей, которые имеют разные уровни приоритета. Браузер, представляя документ пользователю, следует их порядку (как каскаду), последовательно интерпретируя информацию стилей, имеющих разные приоритеты.

Сначала будут проверены и применены стили ***внешнего*** CSS-файла, но если позже будут найдены соответствующие ***встроенные*** стили, то стиль соответствующих элементов будет переопределен, а, если, еще позже, найдется стиль ***прямо внедренный в тег***, то в результате «сработает» именно он.

Итак, в CSS работают следующие правила приоритетов:

— самый низкий приоритет имеют стили браузера по умолчанию;

— затем те, что лежат во внешнем подключенном файле;

— затем те, что встроили внутрь HTML-документа с помощью тега <style>;

— затем те, которые указаны внутри тегов, как их атрибуты;

— самый высокий приоритет у правил с меткой «!important»;

— при равенстве приоритетов будут применены те стили, которые объявлены последними.

Внутри одной таблицы CSS (внешней или встроенной) существует три основных способа использования CSS:

— применение стиля к тегу;

— применение стиля к некоторому ID (идентификатору элемента);

— применение стиля к классу элемента.

Применение стиля к тегу. Если требуется, чтобы все абзацы были записаны через две строки и зеленым цветом, в *CSS* можно добавить следующее объявление:

```
p {
  line-height: 2em;
  color: green;
}
```

Теперь любой контент, заключенный в теги <p>, будет записываться через две строки зеленым цветом.

Переопределение ID. Для любого тега можно указать идентификатор — атрибут id. Каждый id может использоваться на странице *только один раз*, то есть он указывает на некоторый уникальный блок или тег, который должен иметь уникальный (отличный от всех других элементов на этой странице) стиль. В таблице стилей перед ID обязательно указывается знак #.

Например, если вы хотите, чтобы только один абзац текста выводился через два интервала и зеленым цветом, задайте для него ID:

```
<p id = "id1">
  Контент абзаца
</p>
```

И затем примените к нему правило *CSS* следующим образом:

```
#id1 {
  line-height: 2em;
  color: green;
}
```

Переопределение класса. Классы похожи на ID, за исключением того, что на каждой странице можно иметь несколько элементов одного класса. Перед именем класса в таблице стилей обязательно используется символ точка (.).

Если требуется использовать двойной интервал и выделение цветом для двух первых абзацев на странице, то можно будет добавить к ним классы следующим образом:

```
<p class = "id1">Контент первого абзаца </p>
<p class = "id1">Контент второго абзаца </p>
```

И затем применим к ним правило *CSS* следующим образом:

```
.id1 {
  line-height: 2em;
  color: green;
}
```

Описание идентификатора имеет более высокий приоритет по сравнению с описанием класса, приоритет которого, в свою очередь, выше, чем у описания обычного тега. Здесь абзац с идентификатором «food» появится на зеленом фоне.

```
<p id = "food">...</p>

#food {
  background: green;
}

p {
  background: orange;
}
```

В Интернете существует множество источников с удобно структурированной информацией о CSS, которые позволят быстро получить сведения о том, как изменять *оформление* элементов документа[1]. Однако, на наш взгляд, стоит более подробно изложить материал о том, как с использованием CSS можно влиять на структуру страниц.

3.3. Особенности применения CSS для указания формы и расположения блоков

В отличии от строгих привязок стандартных HTML-тегов к своему содержимому (например, тег <p> — абзац, <a> — ссылка), тег <div> является нейтральным. То есть в нем может содержаться разная информация: абзацы, ссылки, картинки, таблицы и т. д. Это тег позволяет описывать определенную функциональную область страницы (область заголовка, область меню, область основного контента), или внутри этих крупных функциональных блоков описывать «подблоки», например, рекламный блок в области заголовка.

---

1 См. напр.: https://www.w3.org/Style/CSS/

Разделение содержимого страницы на блоки позволяет более эффективно управлять его размещением на странице за счет указания стилей каждого блока в каскадных таблицах стилей.

Для начала рассмотрим основные описания CSS, которые касаются формы блока <div>. Следует отметить, что все эти стили можно отнести ко многим другим элементам (картинкам, абзацам и т. п.).

На рис. 3.1 представлены основные свойства, задающие форму блока.

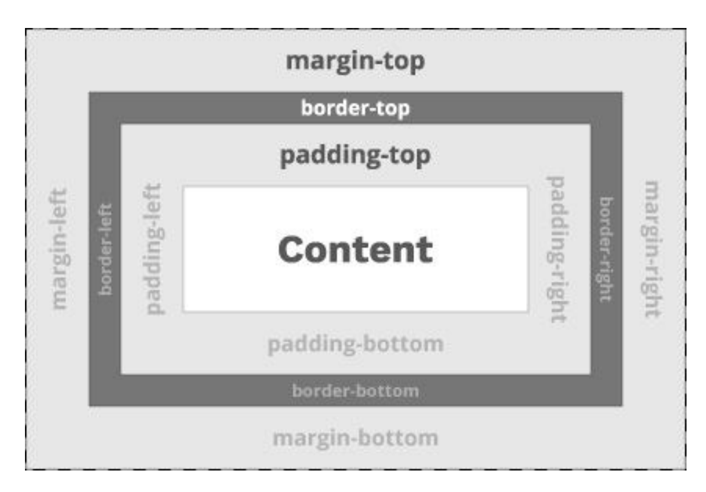


Рис. 3.1. **Основные свойства CSS, задающие форму блока**

Свойство margin устанавливает величину отступа от каждого края элемента. Отступом является пространство от границы этого элемента до внутренней границы его родительского элемента:

```
.element {
  margin: 10px 20px 30px 70px;
}
```

Для любого элемента, который описан с использованием атрибута class = "element", будут выполнены отступы от края экрана (или края родительского для него элемента): сверху — 10 пикселей, справа — 20, снизу — 30 и слева — 70 пикселей (от верхнего края по часовой стрелке). Если указан только один параметр, то будет одинаковый отступ со всех четырех сторон. Если указаны только два параметра, то первый из них показывает величину отступа сверху и снизу, а второй — слева и справа.

Можно указать также конкретный параметр, например,

```
margin-top: 40px;
```

Можно указать также:

```
.element {
  margin: auto;
}
```

Тогда элемент будет отцентрирован по горизонтали относительно родительского блока.

Свойство padding задает величину отступа от границы блока до его содержимого. Используется аналогично свойству margin.

Свойство border задает параметры рамки блока.

Примеры:

```
.element {
  border-color: red green blue orange;
  border-style: solid;
}
```

Рамка будет сплошной линией разного цвета со всех четырех сторон.

```
.element {
  border: 2px solid #e9385a;
  border-radius: 5px;
}
```

Рамка будет с закругленными углами, толщиной 2 пикселя.

Далее рассмотрим свойства, отвечающие за расположение блоков друг относительно друга.

Свойство display. Элементы HTML делятся на две основные категории: блочные и встроенные элементы. *Блочные элементы* (<div>, <p>, <h1> и т. д.) всегда занимают все возможное пространство по горизонтали и начинаются с новой строки. *Встроенные* элементы (<img>, <a> и т. д.) занимают только необходимое пространство. Им не нужно начинаться с новой строки. Свойство display позволяет указать, какой из этих типов должен применяться к описываемому элементу. Примеры:

```
.element {
  display: inline;
}
```

Все элементы класса element (если их несколько подряд) будут расположены в одну строку.

```
.element {
  display: block;
}
```

Каждый элемент класса element займет все возможное горизонтальное пространство, каждый будет выводиться с новой строки.

```
.element {
  display: inline-block;
}
```

Все элементы этого класса будут выстроены в строку, но при изменении размера экрана или изменении значений других атрибутов элемента, будут перестраиваться, например, переходить в следующую строку.

Свойство float. Позволяет указать как другие элементы должны «обтекать» данный элемент, например:

```
.element {
  float: left;
}
.other-box {
  clear: left;
}
```

Все элементы класса element будут прижиматься влево, а следующие за ними элементы будут обтекать их справа. Однако, если встретится элемент класса other-box, он начнет выводится с новой строки, так как свойство **clear: left** «обнуляет» предыдущие float.

То есть становится понятно, что современный подход к описанию web-страниц состоит в том, чтобы в теле документа (body) описывать только содержательные блоки. А то, как они будут размещены в окне браузера, каким цветом, размером и выравниванием будут оформлены — задается с использованием стилей. Современный стандарт разделения содержимого и оформления документа предполагает, что необходимо вынесение стилей в отдельный документ с расширением css и его подключение к HTML-странице с использованием тега <link…>

Ниже представлен пример форматирования страницы с использованием внешней таблицы стилей.

Файл 1.html:

```
<!DOCTYPE html>
  <!– Заголовок документа -->
  <head>
<!– Подключение внешней таблицы стилей -->
    <link href = "style.css" rel = "stylesheet" type = "text/css">

  </head>
  <!-- Начало тела документа -->
  <body>
    <!-- Начало блока содержимого документа -->
    <div class = "container">
      <!-- Начало блока заголовка -->
      <header>
        <h1>Заголовок страницы</h1>
      </header>

      <!-- Начало блока меню -->
      <nav>
        <h1>Меню</h1>
          <!-- Начало списка пунктов меню -->
          <ul>
            <li><a href = "#">Выбор 1</a></li>
            <li><a href = "#">Выбор 2</a></li>
            <li><a href = "#">Выбор 3</a></li>
          <!-- Конец списка пунктов меню -->
          </ul>
        <!-- Конец блока меню -->
        </nav>

      <!-- Начало блока статей -->
      <article>
        <h1>Статья</h1>
        <p>Здесь может быть текст статьи:
         <!-- Вставка картинки -->
        <img src = "1.jpg" style = "height: 50px;">
        </p>
      <!-- Конец блока статей -->
      </article>

      <!-- Начало блока подвала -->
        <footer>
          Это "подвал" страницы
      <!-- Конец блока подвала -->
      </footer>
    <!-- Конец блока документа-->
    </div>
  </body>
</html>
```

Файл style.css:

```
/*-- стиль для блока div с идентификатором container, ширина –
73% от ширины экрана, рамка отсутствует, за счет отступа (margin
auto) – выравнивание блока по центру --*/
div#container {
  width: 73%;
  border: none;
  margin: auto;
}

/*-- стиль для header и footer – заголовок и подвал страницы.
Отступ текста от края блока – 1em (текущий размер шрифта, для
увеличения или уменьшения, можно брать любые пропорции от теку-
щего: 2em, 0.5em); clear: left - отменяет обтекание с левого края
элемента, все другие элементы на этой стороне будут располагаться
под текущим элементом --*/
header {
  padding: 1em;
  color: white;
  background-color: #007196;
  clear: left;
  text-align: center;
}

footer {
  position: absolute;
  left: 0;
  right: 0;
  bottom: 0;
  height: 50px;
  color:white;
  background-color: #007196;
  text-align: center;
}

/*-- для блока меню (nav) float:right – прижимает к правому краю,
остальные элементы обтекают его по левой стороне --*/
nav {
  float: right;
  max-width: 160px;
  height: 164px;
  padding: 1em;
  border: solid #333
}

/*-- стиль только для маркированного списка внутри блока nav,
list-style-type: none  - задает отсутствие маркеров --*/
nav ul {
  list-style-type: none;
  padding: 0;
}
```

```
 /*-- стиль только для гиперссылок внутри элементов списка внутри
блока nav. Стиль ссылок задается по умолчанию --*/
nav ul a {
    text-decoration: none;
}

/*-- стиль блока статей --*/
article {
  padding: 1em;
}
```

При просмотре файла 1.html браузером вы сможете увидеть результат, показанный на рис. 3.2.

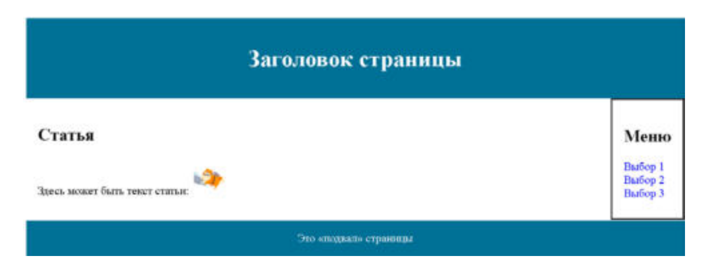


Рис. 3.2. **Результат просмотра документа в браузере**

Контрольные вопросы

1. Для чего применяется технология CSS?
2. Почему используется термин «Каскадные» при определении таблиц стилей?
3. К каким элементам могут быть привязаны стили?
4. Какие правила приоритетности в применении CSS вам известны?
5. Какие свойства используются для указания формы и оформления блоков?

Контрольные задания

1. Создайте страницу с тремя блоками (теги div) содержащими различные абзацы текста. Сохраните файл с именем Блоки.html.
2. Создайте файл с именем Style.css, который позволяет оформить все абзацы текста следующим образом. Цвет шрифта — красный, цвет фона — голубой, рамка — черная, толщиной 3 px, расстояние от текста абзаца до границы родительского блока — 10 px со всех сторон. Подключите эту таблицу стилей к файлу Блоки.html.
3. Измените таблицу стилей так, чтобы один из абзацев был оформлен другим стилем (используйте id для тега p).

4. Измените таблицу стилей так, чтобы остальные два из трех абзацев были оформлены другим стилем (используйте class для двух тегов p).

5. Измените таблицу стилей так, чтобы абзацы выводились по два в ряд (первый и второй на одном уровне, третий — ниже). Используйте стили обтекания для тегов div.

Тема 4
ТЕХНОЛОГИИ АДАПТИВНОЙ ВЕРСТКИ САЙТОВ

После изучения данного раздела студент должен:

знать

- сущность проблемы адаптивной верстки,
- основные виды устройств для просмотра интернет-контента,
- существующие подходы к решению проблемы адаптивности сайтов,
- понятие CSS-фреймворков,
- основные возможности фреймворка Bootstrap,
- назначение технологий сетки и Flexbox, особенности их применения;

уметь

- применять возможности CSS-фреймворка при создании адаптивных сайтов,
- использовать и настраивать готовые компоненты библиотеки,
- описывать структуру страницы с применением средств фреймворка;

владеть

- навыками подключения фреймворка,
- использования готовых шаблонов и компонентов фреймворка для оптимизации процесса верстки сайта,
- переопределения встроенных стилей компонентов фреймворка,
- расположения блоков страницы с применением технологии Grid и Flexbox.

4.1. Понятие и назначение адаптивной верстки

Верстка веб-страниц — это создание структуры гипертекстового документа на основе HTML-разметки с использованием таблиц стилей и клиентских сценариев, таким образом, чтобы элементы дизайна выглядели аналогично макету, который разработал web-дизайнер.

Адаптивная верстка — это подход, предполагающий изменение дизайна в зависимости от поведения пользователя, размера экрана, платформы и ориентации устройства. Другими словами, страница должна автоматически подстраиваться под разрешение экрана, изменять расположение блоков, размер картинок и т. д. Можно разбить устройства на разные категории и верстать для каждой из них отдельно, но это займет слишком много времени, особенно учитывая, что в настоящее время существует целых спектр разнообразных устройств (табл. 4.1).

Таблица 4.1

Основные размеры экранов современных компьютеров

Устройство	Диагональ (в дюймах)	Разрешение (в пикселях)
Смартфоны	2,8—6	240×320 ÷ 2560×1440
Электронные книги	6—8	600×800 ÷ 758×1024
Планшеты	7—13,3	800×480 ÷ 2560×1600
Ноутбуки	10,1—18,4	1280×800 ÷ 3840×2160
Настольные компьютеры	15—55	1024×768 ÷ 3840×2160

В соответствии с современными стандартами, предлагается устанавливать 4 варианта отображения страниц на дисплеях в зависимости от количества экранных пикселей по горизонтали:

— xs (*extra small devices*) — менее 768 пикселей (для смартфонов);

— sm (*small devices*) — от 768 до 991 пикселей (для планшетов);

— md (*medium devices*) — от 992 до 1199 пикселей (для ноутбуков);

— ld (*large devices*) — от 1200 пикселей (для десктопов).

Учитывая распространение в последние годы смарт-часов и инфо-браслетов, необходимо выделять еще один экранный класс устройств с разрешением дисплеев от 240 до 320 пикселей.

Чтобы сайт корректно отображался на устройствах с экранами разных размеров и ориентаций, были разработаны следующие технологии.

Простейший способ — это задавать относительный (в процентах) размер картинок и других элементов страницы.

Пример

Пусть имеется такое описание некоторой галереи изображений на языке HTML.

```
<div class = "image_gallery">
  <div>
    <img src = "../images/1.jpg" title = "Лето" />
  </div>
  <div>
    <img src = "../images/2.jpg" title = "Зима" />
  </div>
  <div>
    <img src = "../images/3.jpg" title = "Осень" />
  </div>
</div>
```

То есть мы добавили несколько изображений и поместили все в один div класса image_gallery.

Теперь рассмотрим фрагмент таблицы стилей для этой галереи.

```
div.image_gallery {
  margin: 0 auto;
  width: 1000px;
  text-align: center;
  max-width: 90%; /*-- контейнер не превышает 90% ширины экрана */
  min-width: 500px;
}

img {
  float: left;
  max-width: 30%; /*-- каждое изображение не превышает 30% ширины
галереи */
  height: auto;
  padding: 2%; /*-- небольшие отступы для изображений --*/
}
```

Каждое изображение в галерее теперь будет изменять свой размер в зависимости от изменения размеров контейнера.

Более сложный способ — чтобы не растягивать и не сужать картинку, ее заменяют ее урезанным фрагментом. Но для того, чтобы переход к этому фрагменту выглядел плавным, основная картинка «подкладывается» как фон в некоторый блок.

Еще одна техника — для маленьких экранов использовать не просто небольшое изображение, но и с меньшим разрешением. Подготавливаются две картинки и указывается тег:

```
<img src = "smallRes.jpg" data-fullsrc = "largeRes.jpg">
```

Если же говорить о современных технологиях адаптивной верстки в целом, то существуют два основных направления.

Первое из них состоит в использовании ***сетки*** (***Grid***), которая позволяет условно разделять горизонтальное пространство экрана на определенное количество колонок (обычно 12), а затем, «вписывать» блоки в эти колонки, при необходимости объединяя их. При этом, если страница имеет более сложную структуру, в которой требуется не только горизонтальное, но и вертикальное выравнивание блоков, приходится использовать дополнительные, не всегда простые методы CSS.

Второе направление предлагает использовать спецификацию CSS ***Flexible Box Layout Module***. Основное преимущество этой технологии верстки состоит в том, что все блоки очень легко делаются «резиновым», что уже следует из названия «flex». Элементы могут сжиматься и растягиваться по заданным правилам, занимая нужное пространство не только по горизонтали, но и по вертикали страницы.

Подробнее применение каждой из этих технологий поясняется в практической части курса.

4.2. CSS-фреймворки

Фреймворк (термин происходит от *framework* — остов, каркас, структура) — класс программного обеспечения, определяющего структуру разрабатываемой программной системы, облегчающего разработку и объединение разных компонентов большого программного проекта.

Часто понятие *фреймворк* употребляют как синоним слова *библиотека подпрограмм*. Однако это более широкое понятие. В отличие от библиотек, фреймворки не просто предоставляют разработчику фрагменты готовых отлаженных функциональных модулей, но и влияют на структуру проекта, определяют способ взаимодействия его составных частей.

CSS-фреймворк — это с одной стороны, набор готовых функциональных блоков, которые объединяют HTML-код и соответствующую таблицу стилей, а с другой — некоторая идеология верстки. CSS-фреймворки созданы для упрощения работы верстальщика, быстроты разработки и исключения максимально возможного числа ошибок верстки, например, проблем совместимости различных версий браузеров и т. д.

При использовании фреймворков CSS-библиотеки, обычно имеющие вид внешнего CSS-файла, «подключаются» к проекту (добавляются в заголовок web-страницы).

В настоящее время используется множество CSS-фреймворков, в том числе Semantic UI, Foundation, Sceleton, но одним из наиболее распространенных CSS-фреймворков сейчас является ***Bootstrap***, который и будет рассмотрен ниже.

4.3. Использование Bootstrap

Комплект всех необходимых модулей фреймворка Bootstrap можно скачать с сайта https://getbootstrap.com/. Установив фреймворк в текущий проект, можно далее продолжать разработку локально. Можно также подключить библиотеку по так называемому ***CDN*** (сети доставки и дистрибьюции контента) и подгружать нужные функции по мере необходимости.

При скачивании Bootstrap вы обратите внимание на то, что создается не только папка с CSS-файлами стилей, но и папка JS-скриптов. Это объясняется тем, что многие из компонентов требуют использования программного кода на языке JavaScript, поэтому часто Bootstrap называют CSS и JS-фреймворком. Подробнее о назначении и использовании JavaScript мы поговорим в следующем разделе.

На следующем шаге рекомендуется скачать с сайта базовый шаблон страницы, который уже содержит все необходимые строки подключения библиотек.

Далее с этого же сайта скачиваются нужные на странице компоненты: меню, списки, компоненты поиска, линейки прокрутки, элементы формы и т. д. Каждый из них представляет собой блок <div>, который описан на языке HTML и имеет в подключенной библиотеке уже готовый набор стилей (и JavaScript-функций). Разработчику нужно только поместить скачанный HTML-код этого элемента в нужное место страницы и подправить стиль (через его переопределение в собственной таблице стилей), если это требуется в соответствии с дизайном сайта.

При этом проблем остается только две:

2) как поместить элемент именно в нужное место страницы;

3) что должно происходить с элементами страницы, если размер экрана будет изменен.

Решить первую проблему позволяют технологии сетки (Grid) или Flexbox, а вторая проблема решается в Bootstrap через так называемые *медиазапросы*. Рассмотрим эти технологии более подробно.

Сетка (Grid System) Bootstrap позволяет горизонтально разделить страницу на 12 столбцов. Если использовать все 12 частей нет необходимости, такие блоки можно легко группировать, создавая более широкие столбцы. Они будут состоять из двух, трех и так далее частей страницы, условно разделенной на 12 частей. Это около 8,3 % ширины всего экрана на столбец. Таким способом можно разделить не только страницу, но и любой блочный элемент, размеры подблоков будут рассчитаны как часть родительского элемента.

В Grid-верстке Bootstrap есть уже упомянутые выше четыре основных класса устройств: xs, sm, md и lg. Удобство сетки состоит в том, что эти классы можно комбинировать, чтобы создавать адаптивную верстку и точно знать, как макет будет отображаться при том или ином размере экрана.

Базовая структура макета сетки в Bootstrap выглядит так:

```
<!-- Выделяется первый горизонтальный блок -->
<div class = "row">
<!-- Внутри него выделяется вертикальный блок -->
  <div class = "col-*"></div>
</div>

<!-- Выделяется второй горизонтальный блок -->
<div class = "row">
  <!-- Внутри него выделяются три вертикальных блока -->
  <div class = "col-*"></div>
  <div class = "col-*"></div>
  <div class = "col-*"></div>
</div>

<div class = "row">
  ...
</div>
```

На основе базового макета создается любой другой. Описывается строка с ячейками: <div class = "row"> и в нее добавляется нужное количество столбцов с соответствующими классами .col-*.

Обратите внимание, что сумма ячеек для каждого типа (sm, xs, md, lg) в ячейке .col-* не должна превышать 12 частей для каждой строки.

Следующий пример показывает, как сверстать три равных по ширине колонки, которые будут отображаться горизонтально на планшетах и более крупных экранах. На экране мобильного такая верстка будет растянута и каждая из ячеек займет всю ширину экрана (если ширина всех колонок одинакова, количество частей в колонке можно не указывать):

```
<div class = "row">
  <div class = "col"> Содержимое первой колонки</div>
  <div class = "col"> Содержимое второй колонки </div>
  <div class = "col"> Содержимое третьей колонки </div>
</div>
```

В следующем примере страница делится на две колонки разной ширины, которые будут отображаться горизонтально на любых устройствах, кроме мобильных телефонов:

```
<div class = "row">
  <div class = "col-4"> Содержимое первой колонки </div>
  <div class = "col-8"> Содержимое второй колонки </div>
</div>
```

Еще один пример: две колонки с двумя вложенными колонками. В этом случае страница будет разделена на две разные колонки на планшетах и больших экранах, а большая колонка в свою очередь будет разделена на две равные колонки. На экранах мобильных телефонов эти колонки будут растягиваться на всю ширину экрана.

```
<div class = "row">
  <div class = "col-8"> Содержимое первой колонки
    <div class = "row">
      <div class = "col-6"> Содержимое первой встроенной колонки
</div>
      <div class = "col-6"> Содержимое второй встроенной колонки
</div>
    </div>
  </div>
  <div class = "col-4"> Содержимое второй колонки </div>
</div>
```

Таким образом легко можно контролировать, как отображать ячейки из шаблона, а верстка становится мобильной без дополнительных усилий.

Однако для того, чтобы контролировать размещение блоков также по вертикали, используется так называемая *решетка*,

которая реализована только в версии Bootstrap 4 с помощью технологии ***Flexbox.***

Рассмотрим пример галереи на Flexbox[1]. Пусть необходимо вывести 10 изображений. Такой блок на HTML можно описать следующим образом:

```
<main class = "gallery">
  <img src = "/img1.jpg">
  <img src = "/img2.jpg">
  <img src = "/img3.jpg">
  <img src = "/img4.jpg">
  <img src = "/img5.jpg">
  <img src = "/img6.jpg">
  <img src = "/img7.jpg">
  <img src = "/img8.jpg">
  <img src = "/img9.jpg">
  <img src = "/img10.jpg">
</main>
```

В этом случае CSS с Flexbox позволяет сделать следующее:

```
gallery {
  /* Галерея займет все свободное пространство экрана */
  min-height: 100vh;

  /* Все изображения уменьшатся, чтобы уместиться в одну линию */
  display: flex;

  /* Разрешить переносить не поместившиеся элементы на следующую
строку */
  flex-wrap: wrap;

  /* Сохранять исходный размер изображений */
  align-items: flex-start;

  /* Разместить блоки по центру по горизонтали */
  justify-content:center;

  /* Разместить блоки по центру по вертикали */
  align:center;
}
```

Очевидно, что подобные, и гораздо более сложные настройки можно выполнять с использованием свойств Flexbox над любыми блоками[2].

---

1 https://keynikell.ru/

2 Подробнее см.: https://tproger.ru/translations/how-css-flexbox-works/

4.4. Использование медиазапросов

Медиазапросы позволяют выполнять определенные действия со страницей при изменении ее размера.

Пример медиа-запроса:

```
@media screen and (max-width: 768px)
{
  body {
    font-size: 9pt;
  }
}
```

Данный код означает следующее: «Если ширина окна браузера станет меньше 768 px, то применить стили, указанные в фигурных скобках».

Пусть имеется следующий HTML-документ:

```
<!DOCTYPE html>
<head>
  <title>Медиа-запросы</title>
  <style>
    body {
      font-size: 15pt;
    }
    @media screen and (max-width: 768px) {
      body {
        font-size: 9pt;
      }
    }
  </style>
</head>
<body>
  <p>Текст</p>
</body>
</html>
```

Если открыть эту страницу в браузере и начать уменьшать его размер, при достижении размера 768 пикселей текст уменьшится до 9 pt.

В медиазапросах могут указываться и другие параметры, такие как min-width, max-height и min-height, которые будут срабатывать при указанной ширине и высоте. Также можно комбинировать разные параметры через and:

```
@media screen and (max-width: 768px) and (max-height: 300px) {
  body {
```

```
    font-size: 9pt;
  }
}
```

В данном случае стили будут подключаться только при ширине, меньшей либо равной 768 px, и при высоте, меньшей либо равной 300 px.

Контрольные вопросы

1. Определите понятия «верстка» и «адаптивная верстка» web-страниц.
2. Какие подходы, используемые при адаптивной верстке, вам известны?
3. Что такое фреймворк? Чем фреймворки отличаются от библиотек подпрограмм?
4. Что такое CSS-фреймворк? Какие CSS-фреймворки вам известны?
5. Каковы основные возможности Bootstrap?
6. Как верстается страница при применении Bootstrap?
7. Поясните основные принципы использования Grid System в Bootstrap.
8. Для чего используются медиазапросы?

Контрольные задания

1. Создайте две страницы HTML на основе базового документа (Starter template) с сайта bootstrap https://getbootstrap.com/docs/4.5/getting-started/introduction/
2. Проверьте подключение Bootstrap по технологии CDN. Вы должны увидеть сообщение Hello World! при запуске каждой из страниц.
3. Разместите внутри области body каждой из страниц сетку Bootstrap, содержащую одну строку, внутри которой две колонки. Первая колонка занимает одну треть экрана, вторая — две трети.
4. Разместите на обеих страницах в более узкой колонке компонент вертикального меню, скачанный со страницы https://getbootstrap.com/docs/4.4/components/navs/. Настройте меню так, чтобы оно позволяло переключаться между двумя этими страницами.
5. Поместите в более широкую колонку каждой из страниц два разных компонента Bootstrap, позволяющие продемонстрировать возможности фреймворка.

Тема 5
ОСНОВЫ ПРИМЕНЕНИЯ JAVASCRIPT

После изучения данного раздела студент должен:

знать

- особенности клиент-серверного взаимодействия на основе протокола HTTP,
- возможности клиентской обработки web-страниц,
- назначение, возможности и ограничения языка JavaScript, его библиотек и фреймворков,
- основы создания и отладки программного кода на языке JavaScript,
- возможности и преимущества JQuery и React;

уметь

- создавать клиентские сценарии для обработки web-страниц с применением «чистого» JavaScript, а также с применением современных JavaScript библиотек и фреймворков;

владеть

- навыками создания и отладки программного кода на языке JavaScript,
- структурирования кода и применения функций,
- подключения библиотек и фреймворкаов,
- использования готовых шаблонов и компонентов фреймворков для оптимизации разработки клиентских скриптов.

5.1. Назначение и возможности скриптовых языков программирования

Как вам уже известно, в системе WWW *клиент-браузер* посылает HTTP-запросы некоторой программе, расположенной на удаленном компьютере в сети Интернет, которая выполняет функции *сервера*, обрабатывает запрос и отправляет обратно клиенту ответ в виде, как правило, HTML-страницы. Таким образом, в основе работы web-приложения лежит так называемая **модель взаимодействия клиент-сервер,** которая позво-

ляет разделять функциональность и вычислительную нагрузку между клиентскими приложениями (заказчиками услуг) и серверными приложениями (поставщиками услуг).

Результатом работы web-приложения является web-страница, отображаемая в окне браузера. При ее формировании, в рамках клиент-серверной модели взаимодействия, функции web-приложения могут выполняться как на компьютере клиента, так и на компьютере сервера.

Одним из типов приложений, предназначенных для выполнения на стороне клиента, являются сценарии, написанные на специальных скриптовых языках программирования: JavaScript, VBScript и др. Исходный текст сценария представляет собой часть web-страницы или внешний подключаемый файл, поэтому сценарий JavaScript передается клиенту вместе с документом, в состав которого он входит. Обрабатывая HTML-документ, браузер обнаруживает исходный текст сценария и выполняет его.

JavaScript является наиболее популярным языком сценариев в Интернете и может быть обработан большинством современных браузеров. Он является интерпретируемым языком (это значит, что скрипты исполняются без предварительной компиляции) и распространяется свободно, без необходимости покупки лицензии.

JavaScript может использоваться для управления web-страницами, взаимодействия с пользователем и взаимодействия с web-сервером.

Так, с использованием JavaScript можно:

— добавлять новый HTML-код на страницу, изменять существующее содержимое, модифицировать стили;

— обрабатывать действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш;

— отправлять сетевые запросы на удаленные серверы, скачивать и загружать файлы;

— сохранять и получать информацию на компьютере клиента (cookies).

При этом возможности JavaScript в браузере ограничены для обеспечения безопасности пользователя. Цель этого — предотвращение доступа недобросовестной web-страницы к личной информации или нанесения ущерба данным пользователя. Ниже приведены примеры таких ограничений:

— JavaScript не может использоваться для получения или сохранения произвольной информации на диске компьютера-клиента, копирования информации, запуска на выполнения других программ;

— JavaScript имеет ограниченные возможности работы с файлами;

— существуют способы взаимодействия JavaScript с камерой/микрофоном и другими устройствами, но они требуют явного разрешения пользователя;

— JavaScript не имеет возможностей для обращения со страницы, открытой в одной вкладке браузера, к другой вкладке браузера, если они были загружены с разных сайтов. Чтобы обойти это ограничение, обе страницы должны согласиться с этим и содержать JavaScript-код, который специальным образом обменивается данными.

5.2. Основы создания скриптов на языке JavaScript

Изучение JavaScript начнем, как это принято, с вывода приветствия.

Пример ниже реализует вывод приветствия в модальном окне:

```
<!-- документ, содержащий скрипт -->
<!DOCTYPE html>
  <body>
    <!-- начало скрипта -->
    <script>
      /* Инструкция JScript, которая выведет
         сообщение "Hello, world!" */
      alert("Hello, world!");
    // Конец скрипта
    </script>
  </body>
</html>
```

Данный пример показывает несколько правил оформления скриптов:

— каждая инструкция заканчивается точкой с запятой;

— однострочные комментарии начинаются с двойной косой черты //.

— многострочные комментарии ограничиваются знаками /*…*/.

Для внедрения JavaScript непосредственно в текст HTML-страницы используется тег HTML ***<script>…</script>***.

JavaScript также может быть помещен во внешний файл. Внешние файлы JavaScript обычно содержат код, который используется несколькими различными web-страницами. Внешние файлы JavaScript имеют расширение .js. Внешний скрипт не должен содержать теги <script>…</script>. Для вызова внешнего скрипта из текста документа используется тег <script> следующим образом.

```
<!DOCTYPE html>
  <head>
...
    <script src = “js/MyScript.js”></script>
  </head>
  <body>
...
  </body>
</html>
```

В следующих главах мы рассмотрим особенности языка JavaScript и его применения для клиентской обработки в web-приложениях.

5.3. Основы языка JavaScript

Переменные и ***операторы*** JavaScript используются для хранения значений или выражений и выполнения простых операций над ними.

Переменная может иметь короткое имя, например, *x*, или более информативное имя, например, *carName* (название автомобиля). Имена переменных чувствительны к регистру (y и Y — это две разных переменных). Имена переменных должны начинаться с буквы или символа подчеркивания.

Создание переменных в JavaScript также называют «объявлением» переменных. Объявление переменных JavaScript выполняется с помощью ключевого слова **let**. После создания перемененной в нее можно поместить какое-либо значение. Далее это значение можно тем или иным образом использовать:

```
let fioStudent; // объявляем переменную
fioStudent = 'Иванов'; // присваиваем ей значение
alert(fioStudent); // выводим содержимое переменной
```

JavaScript является языком программирования с динамической типизацией. Это означает, что при присвоении значений переменным их тип определяется автоматически. В данном вводном курсе мы рассмотрим только самые распространенные из этих типов[1].

Первый тип — ***строковые переменные***. Они задаются одним из следующих способов:

```
/* Можно использовать двойные ПРЯМЫЕ кавычки */
let str1 = "привет!";
/* Можно использовать одинарные ПРЯМЫЕ кавычки */
let str2 = 'привет!';
/* Обратные кавычки позволят встраивать значения других переменных
или выражений*/
let phrase = `Вася ${str2}`;

// выведет: Вася привет!
alert(phrase);
```

Встраивать в строку можно также выражение, например, так:

```
alert('результат: ${1 + 2}' ); // выведет "результат: 3"
```

То есть выражение нужно заключить в фигурные скобки, перед которыми поставить знак доллара, а строку взять в «обратные» кавычки.

Самый простой оператор для строковых переменных — это сцепление или конкатенация строк (+):

```
let name = 'Вася';
let message = 'привет';
let m = name+ ', ' + message;
```

В результате в переменной m будет лежать значение ‘Вася, привет’.

Второй тип данных — ***числа***.

Этот тип позволяет использовать как целочисленные значения, так и числа с плавающей точкой.

```
let n = 123;
n = 12.345;
```

---

1 Подробнее о типах JavaScript см.: https://learn.javascript.ru/types

Существует множество ***операций для чисел***, например, умножение «*», деление «/», сложение «+», вычитание «-» , остаток от деления нацело «%», увеличение на единицу (инкремент) «++», возведение в степень «**», и т. д.

Третий из базовых типов — ***булев тип*** (boolean). Переменные этого типа могут принимать только два значения: true (истина) и false (ложь). Значения булевого типа обычно получаются как результат операций сравнения (>, <, == (равно), < = (меньше или равно), > =). Например, результатом выражения 2 > 3 будет булево значение false.

Над логическими значениями могут выполняться логические операции. Пусть x = 6 и y = 3. Таблица ниже объясняет логические операции:

Оператор	Описание	Пример
&&	логическое И	(x < 10 && y > 1) - это истина
||	логическое ИЛИ	(x == 5 || y == 5) это ложь
!	логическое НЕ	!(x == y) это истина

Очень часто при взаимодействии с пользователем приходится учитывать различные даты или время. Для того, чтобы их вводить и хранить существует специальный тип ***Date***. Пример:

```
let now = new Date();
/* Создает объект Date с текущей датой и временем */
alert( now );
```

new Date(year, month, date, hours, minutes, seconds, ms) создает дату с указанными параметрами:

```
/* год, месяц (нумерация от нуля), число, часы, минуты, секунды,
миллисекунды */
let date = new Date(2019, 0, 1, 2, 3, 4, 567);
alert( date ); // 1.01.2019, 02:03:04.567
```

Для данных типа Date наиболее часто встречаются операции получения одного из параметров:

— getFullYear() — получить год (из 4 цифр)

— getMonth() — получить месяц, от 0 до 11.

— getDate() — получить число месяца, от 1 до 31.

— getHours(), getMinutes(), getSeconds(), getMilliseconds() — получить соответствующие компоненты.

```
let y = date.getFullYear(); // Получение года
```

При выполнении скрипта инструкции обычно сами преобразуют переданные им значения к нужному типу данных. Например, alert автоматически преобразует любое значение к строке. Математические операторы преобразуют значения к числам. Однако бывают ситуации, когда нужно явно преобразовать значение в нужный тип.

Далее показаны функции, которые преобразуют значения переменных в нужный тип:

```
/* b теперь содержит не строку, а число 123 */
let a = "123";
let b = Number(a);

/* b теперь содержит не число, а строку "123" */
let a = 123;
let b = String(a);

/* b теперь содержит не число 1, а значение true */
let a = 1;
let b = Boolean(a);
```

Инструкции взаимодействия с пользователями. Чтобы показать пользователю результаты работы скрипта или запросить у него какие-либо действия, необходимые для работы скрипта, используются так называемые модальные окна. Они бывают трех типов.

1. ***alert*** — выводит некоторое сообщение, которое исчезает, после нажатия кнопки ОК:

```
alert("Это результат работы скрипта JavaScript");
```

В результате появится окно с соответствующим текстом и кнопкой:

2. ***prompt*** — содержит поле для ввода информации от пользователя и две кнопки, ОК и Отмена:

```
let name = prompt("Введите Ваше имя:","Вася?");
```

В результате пользователь увидит модальное окно, в котором есть область для ввода ответа на вопрос:

Подтвердите действие
Введите Ваше имя:
Вася?
ОК Отмена

В этой области находится значение второго необязательного параметра (значение по умолчанию). Если пользователь нажимает на кнопку ОК, введенное значение помещается в строковую переменную name.

3. ***confirm* —** показывает сообщение и ждет, пока пользователь нажмет ОК или Отмена. В результате возвращается значение true, если нажата ОК, и false, если нажата кнопка «Отмена» или Esc с клавиатуры.

```
let a = confirm("Запомнить пароль?");
```

Подтвердите действие
Запомнить пароль?
ОК Отмена

Оператор условия. В тексте программ на JavaScript часто применяются операторы условия, которые позволяют проверить, возвращается ли значение «истина», а затем выполнить другой код в зависимости от результата. Самая распространенная форма условия называется ***if … else.*** Например:

```
let language = 'JavaScript';
if (language == 'JavaScript') {
  alert('Да, я изучаю JavaScript!');
}
else {
  alert('Нет, JavaScript я не изучаю');
}
```

Выражение внутри if использует оператор тождества (==), чтобы сравнить переменную *language* со строкой JavaScript и увидеть, равны ли они. Если это сравнение возвращает true, выполняется первый блок кода. Если нет, первый блок кода

пропускается и выполняется второй блок кода, после инструкции else.

Для задания более сложных условий выбора используется конструкция ***if...else if...else***:

```
let d = new Date();
let time = d.getHours();
if (time < 10)
{
  alert('Доброе утро');
}
else
{
  if (time >=  10 && time <= 16)
  {
    alert('Добрый день');
  }
  else
  {
    alert('Добрый вечер');
  }
}
```

Циклы. Это еще один тип структурных операторов языка, которые используются для многократного повторения одного и того же блока инструкций. В JavaScript существует два типа циклов.

Цикл ***for*** используется, когда вы знаете заранее, сколько раз должен быть выполнен скрипт или его фрагмент.

Пример ниже определяет цикл, который начинается с i = 0. Цикл продолжает выполняться до тех пор, пока i меньше или равно 5. Значение i будет увеличиваться на 1 при каждой итерации цикла.

```
let i;
for (i = 0; i <= 5; i++) {
  alert("Число i равно " + i);
}
```

Цикл ***while*** выполняет блок кода, пока указанное условие истинно. Пример ниже решает предыдущую задачу с использованием while.

```
let i = 0;
while (i <= 5) {
  alert("Число i равно " + i);
  i++;
}
```

5.4. Функции JavaScript

Чтобы предотвратить выполнение скрипта браузером, пока страница полностью не загрузилась, или чтобы многократно выполнить одно и то же действие в разных частях страницы, рекомендуется помещать скрипт в функцию. Функция содержит код, который будет выполнен в процессе работы со страницей, но только если наступило некоторое событие.

Функции можно поместить:

— в область заголовка документа <head>…</head>, обрамляя их тегами <script>…</script>;

— в конец HTML-страницы, обрамляя их тегами <script>… </script>;

— во внешний JS-файл, подключая его в области заголовка или в конце тела страницы с использованием тегов <script src = "…file.js"></script>

Общий синтаксис определения функции:

```
function имя_функции(var1, var2, ..., varN)
{
  некоторый код
}
```

Параметры var1, var2 и т. д. — это переменные или значения, *передаваемые функции*. Фигурные скобки определяют начало и конец функции, например:

```
function showMessage(name, text) // аргументы: name, text
{
  alert(name+ ', ' + text);
}

showMessage('Аня', 'Привет!'); // результат: вывод в модальном
окне Аня: Привет!
```

Функция без параметров должна включать круглые скобки () после имени функции. То есть, если функция не принимает никаких значений извне, ее нужно использовать так:

```
function showMessage() // аргументов нет
{
  alert('Аня, привет');
}

showMessage(); // результат:Аня, привет!
```

Возврат значений. Предложение ***return*** используется для указания значения, которое возвращается из функции. Таким образом, функции, которые возвращают значения, должны использовать предложение return.

Следующий пример возвращает произведение двух чисел (a и b).

```
function product(a, b)
{
  return a*b;
}

alert(product(4, 3));
```

5.5. Строки и массивы JavaScript

Для создания программ на любом современном языке программирования уже недостаточно просто переменных. Все языки «умеют» работать с более сложными типами данных, которые объединяют информацию в массивы, объекты, перечни, последовательности и т. д., и содержат средства эффективной обработки таких сложных структур. Здесь мы ограничимся только рассмотрением строк и массивов JavaScript.

В JavaScript любые текстовые данные являются строками. Не существует отдельного типа «символ», который есть в ряде других языков. Все символы в строке кодируются в формате UTF-16, независимо от кодировки страницы.

Строка в JavaScript — это объект, который имеет множество полезных свойств и методов.

Свойство ***length*** определяет длину строки:

```
let str = 'Hello, world! ';
alert(str.length); // выведет 13
```

Свойство записывается БЕЗ круглых скобок!

Получить символ, который занимает позицию N, можно с помощью квадратных скобок: **[N]**. Первый символ имеет номер 0.

```
let str = 'Hello';
alert( str[0] ); // получаем первый символ H
```

```
// получаем последний символ
alert( str[str.length - 1] ); // о
```

Содержимое строки в JavaScript нельзя изменить. Изменить строку можно только при ее замене с тем же именем.

```
let str = 'Да';
str[0] = 'д'; // ошибка
str = 'д' + str[1]; // заменяем строку
```

Методы ***toLowerCase*()** и ***toUpperCase*()** меняют регистр символов:

```
alert( 'Hello, World!'.toUpperCase() ); // HELLO, WORLD!
alert( "Hello, World!'.toLowerCase() ); // hello, world
```

Для ***поиска подстроки*** (определения того, есть ли такое включение символов в исходной строке) можно использовать метод ***str.indexOf*(*substr*, *N*).** Он начинает искать substr в строке str, начиная с позиции N, и возвращает позицию, на которой располагается совпадение, либо -1 при отсутствии совпадений.

В следующем примере в строке «абракадабра» выполняется поиск индексов, начиная с которых в ней содержится строка «абра».

```
let str = 'абракадабра';

let target = 'абра'; // цель поиска

let N = 0; // позиция с которой начинаем поиск
while (true) // цикл, который позволит пройти по всей строке
{
  let N1 = str.indexOf(target, N); // сохраняем позицию
  if (N1 == -1) {
    // если не было включений, выходим из цикла
    break;
  }
  else {
    alert( 'Найдено тут: ${N1}' );
    // если было найдено совпадение, выводим позицию
    N = N1 + 1; // продолжаем с позиции, которая следует после
заданной
  }
}
```

В JavaScript есть несколько методов для получения подстроки, например, ***substr*** и ***slice***.

Метод slice позволяет «вытащить» символы из исходной строки от начальной позиции до конечной (не включая символ конечной позиции).

```
let str = "абракадабра";
alert( str.slice(1, 5) ); // ‘брак’, символы от 1 до 5
(не включая 5)
```

Если второй параметр не задан, выбираются все символы до конца строки.

Метод substr позволяет «вытащить» символы из исходной строки от начальной позиции в количестве L.

```
let str = "абракадабра";
alert( str.substr(2, 3) ); // рак, получаем 3 символа,
начиная с позиции 2
```

Задача обработки строковых выражений, которые вводит пользователь на странице является очень распространенной и важной. Поэтому перед началом обработки, во избежание ошибок, необходимо проверить, не является ли строка пустой.

Пример. Пусть необходимо написать функцию, которая переводит первую букву введенной строки в верхний регистр:

```
function ucFirst(str) {
  if (!str) {
    return "Строка пуста";
  }
  else {
    return str[0].toUpperCase() + str.slice(1);
  }
}
let str = "вася";
alert( ucFirst(str) ); // результат: Вася
```

Также, как в любом другом языке высокого уровня, в JavaScript **массивы** (**array**) используются для хранения и обработки наборов каких-либо однородных данных: пользователей, товаров, элементов HTML и т. д.

Для создания нового пустого массива может использоваться конструкция:

```
let arr1 = []; // Пустой массив с именем arr1
let cities = ['Москва', 'Тула', 'Сочи']; // Массив с названиями
городов
```

К отдельным элементам массива обращаются по индексам (первый элемент массива имеет индекс 0):

```
alert(cities[0]); // Результат: Москва
```

Можно изменить любой из элементов массива:

```
cities[2] = 'Калуга'; // Теперь [‘Москва’, ‘Тула’, ‘Калуга’]
```

Можно добавить элемент к имеющемуся массиву:

```
cities[3] = 'Орел'; // Теперь [‘Москва’, ‘Тула’, ‘Калуга’,’Орел’]
```

Количество элементов массива получают через свойство ***length***:

```
alert(cities.length ); // Результат: 4
```

Наиболее распространенный алгоритм работы с массивом — это ***перебор его значений*** в цикле:

```
let cities = ['Москва', 'Тула', 'Сочи'];
for (let i = 0; i < cities.length; i++)
{
  alert( cities[i] );
}
```

5.6. События JavaScript

JavaScript позволяет создавать динамичные web-страницы.

В соответствии с **объектной моделью документа** (*Document Object Model*, DOM), каждый HTML-тег является объектом. Вложенные теги являются «потомками» родительского элемента. Текст, который находится внутри тега, также является объектом. Все эти объекты доступны JavaScript, мы можем использовать их для изменения страницы.

События — это действия, которые JavaScript может обнаружить и отследить. Таким образом, процесс программирования клиентской части web-приложений состоит в описании JavaScript-функций, которые связываются с наступлением различных браузерных событий.

В табл. 5.1 представлены основные события, которые могут быть связаны с web-страницей.

Таблица 5.1

Основные события браузера

Группа	Событие	Описание
События мыши	click	Однократный щелчок левой кнопкой мыши
	contextmenu	Однократный щелчок правой кнопкой мыши
	mouseover и mouseout	Наведение мыши на элемент или отведение от него
	mousemove	Движение мыши
События формы	submit	Пользователь отправил форму
	focus	Пользователь наводит фокус на элемент формы, например, <input>
События документа	DOMContentLoaded	Документ загружен, DOM документа полностью построен
События клавиатуры	keydown и keyup	Нажатие и отпускание клавиши

Событию можно назначить ***обработчик***, то есть функцию, которая сработает, как только событие произошло.

Описание функции-обработчика может выполняться двумя способами.

1 способ. Применяется, если к некоторому событию должен быть применен только один обработчик. Пример:

```
<input type = "button" id = "button1" value = "Кнопка">
<script>
  function mes() {
    alert( 'Привет!' );
  }
  button1.onclick = mes;
</script>
```

Этот код создает кнопку с идентификатором *button1*, событие *onclick* (нажатие) для которой вызовет функцию, выводящую приветствие. Обратите внимание, что функция в скрипте вызывается как обработчик кнопки с идентификатором *button1*, а, значит, вызов этой функции должен быть расположен на web-странице ниже, чем описание этой кнопки. Напри-

мер, все подобные вызовы, можно расположить в скрипте, находящемся перед закрытием body.

2 способ применяется тогда, когда нужно иметь возможность добавлять несколько обработчиков одному событию. Для этого используют метод ***element.addEventListener(event, handler)***, где event — это событие, а handler — функция-обработчик. Например:

```
// Выводим кнопку с идентификатором "button1”
<input id = "button1" type = "button" value = "Нажми меня"/>
// Начинаем скрипт
<script>
  // Описываем первую функцию с некоторыми действиями
  function handler1() {
    alert('Вы нажали кнопку первый раз');
  }
// Описываем вторую функцию с другими действиями
  function handler2() {
    alert('Вы нажали кнопку второй раз!');
  }
// Метод отслеживает событие и каждый раз выполняет следующую
по счету функцию

  button1.addEventListener("click", handler1); // ‘Вы нажали
кнопку первый раз’
  button1.addEventListener("click", handler2); // ‘Вы нажали
кнопку второй раз!'
</script>
```

5.7. Несколько примеров использования JavaScript

В первом примере по данной теме предлагаем создать раскрывающееся меню с применением JavaScript.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset = "utf-8">
    <style>
    /*-- Объявляем стили для будущих тегов и классов меню */
    p{
      margin:0px;
      padding:0px;
    }
    .menu {
      cursor:pointer;
    }
```

```
/* Элементы, имеющие класс submenu, являются скрытыми, с нулевой
высотой, полностью прозрачными, но при их появлении будет исполь-
зоваться плавная анимация, ускоряющаяся к концу */
    .submenu {
      overflow: hidden;
      height: 0;
      opacity: 0;
      transition: all 0.5s ease-in;
    }
    </style>
  </head>
  <body>
    <ul>
      <li class = "menu">
        <p>Новости</p>
          <ul class = "submenu">
            <li >Новости культуры</li>
            <li >Новости спорта</li>
          </ul>
      </li>
      <li class = "menu">
      <p>Наши продукты</p>
        <ul class = "submenu">
          <li >Продукт 1</li>
          <li >Продукт 2</li>
        </ul>
      </li>
    </ul>
    // Описываем скрипт
    <script>
    // Соберем коллекцию из элементов класса menu
    let elMenu = document.getElementsByClassName('menu');
    /* В цикле обойдем эту коллекцию и укажем, что, если с любым
из ее элементов случается событие наведения или отведения мыши,
нужно выполнять одну из описанных ниже функций*/
    for(let i = 0; i<elMenu.length; i++) {
      elMenu[i].addEventListener("mouseover", showSub, false);
      elMenu[i].addEventListener("mouseout", hideSub, false);
    }
    // Функция показа подменю
    //Для всех дочерних элементов, относительно текущего (this)
    function showSub(e) {
      if(this.children.length>1) {
        this.children[1].style.height = "auto";
        this.children[1].style.overflow = "visible";
        this.children[1].style.opacity = "1";
      }
      else {
        return false;
      }
    }
```

```
    // Функция скрытия подменю
    // Для всех дочерних элементов, относительно текущего (this)
    function hideSub(e) {
      if(this.children.length>1) {
        this.children[1].style.height = 0;
        this.children[1].style.overflow = "hidden";
        this.children[1].style.opacity = 0;
      }
      else {
        return false;
      }
    }
    </script>
  </body>
</html>
```

В результате получим примерно такую страницу:

- Новости
  - Новости культуры
  - Новости спорта
- Наши продукты

Пункт верхнего уровня Новости будет «раскрываться» при наведении на него мыши (событие mouseover). При этом пункты подменю плавно становятся видимыми.

Конечно, более правильным будет разнести HTML-код, CSS и JS-код в разные файлы и подключить стили и скрипты к основному HTML-документу как внешние источники. Этот подход демонстрируется в следующем примере, который представляет некоторую минигалерею (рис. 5.2) (использованы материалы сайта https://webformyself.com).

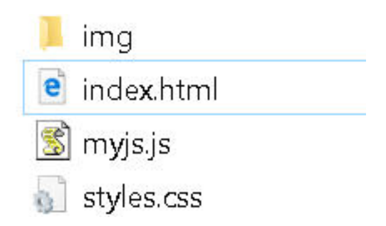






Рис. 5.2. **Результат вывода проектируемой минигалереи**

Этот проект имеет следующую структуру:



Изображения хранятся в папке img, а в корневой папке хранятся основной файл index.html, файл стилей styles.css и скрипт myjs.js.

Предположим, что такую галерею мы хотим реализовать на «чистом» HTML. Тогда содержание файлов index.html и styles.css должно быть примерно следующим.

Файл index.html.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8"/>
    <!-- Подключаем внешний файл стилей -->
    <link rel = "stylesheet" href = "styles.css">
  </head>
  <body>
    <!-- описываем блок со всем содержимым страницы -->
    <div class = "app">
      <!-- Организуем блок, который позволит выравнивать экзем-
пляры галереи с применением flexbox -->
      <div class = "list">
        <!-- Организуем карточки, в которых собрана вся необходи-
мая информация о каждом экземпляре галереи -->
        <div class = "card">
          <div class = "card_img">
            <img src = 'img/330px-Apple-II.jpg'>
            <h3>Apple-II<h3>
          </div>
          <p>год выпуска - 1977</p>
        </div>
        <div class = "card">
          <div class = "card_img">
            <img src = 'img/330px-IBM_5150_PC.jpg'>
            <h3>IBM-5150<h3>
          </div>
          <p>год выпуска - 1982</p>
        </div>
        <div class = "card">
          <div class = "card_img">
            <img src = 'img/330px-Macintosh_128k_transparency.png'>
            <h3>Macintosh_128<h3>
          </div>
```

```
          <p>год выпуска - 1984</p>
        </div>
      </div>
    </div>
  </body>
</html>
```

Файл styles.css.

```
.app {
    max-width: 1000px;
    margin: 50px auto;
}

.list {
    display: flex;
    justify-content: space-between;
}

.card {
    width: 200px;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 3px;
    transition: box-shadow .3s ease-in-out;
}

.card:hover {
    box-shadow: 2px 2px 0 rgba(0, 0, 0, .15);
}

.card-img {
    display: flex;
    align-items: center;
    height: 150px;
}
```

Вы можете заметить, что если возникнет необходимость вывести в этой галерее не 3, а 103 объекта, повторение этих «карточек» в коде HTML становится довольно неприятной рутинной задачей. Попробуем решить ее более эффективно с использованием JavaScript.

В подготовленном в папке проекта файле myjs.js введем следующий код:

```
/* Формируем структуру, описывающую данные галереи */
let comps = [
  {pict:'img/330px-Apple-II.jpg',
    name:'Apple-II',
    note:'год выпуска - 1977'},
```

```
  {pict:'img/330px-IBM_5150_PC.jpg',
    name:'IBM-5150',
    note:'год выпуска - 1982'},
  {pict:'img/330px-Macintosh_128k_transparency.png',
    name:'Macintosh_128',
    note:'год выпуска - 1984'}
 ]
/* Создаем функцию, получающую в качестве аргумента один из эле-
ментов этой структуры. Конкретные значения выводимых элементов
заменяем на свойства переменной, соответствующие элементам описан-
ной выше структуры */
/* Важно использовать для вывода HTML-КОДА КОСЫЕ КАВЫЧКИ, ИМЕННО
ТАК, как показано ниже*/
function createCard(comp){
  return `
    <div class="card">
      <div class="card_img">
        <img src="${comp.pict}"/>
        <h3>${comp.name}</h3>
      </div>
      <p>${comp.note}</p>
    </div>
  `
}
/* Получаем набор данных (список, которые содержит все элементы
структуры, оформленные в соответствии с функцией CreateCard)*/
let ListComp = comps.map(comp = > createCard(comp))

/* Разделяем полученный список элементов, например через пробел */
let html = ListComp.join(' ')

// Выводим разделенный список на странице в область с классом list
document.querySelector('.list').innerHTML = html
```

При этом, естественно, в файле index.html нужно убрать все содержимое документа, и подключить скриптовый файл:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8"/>

    <link rel = "stylesheet" href = "styles.css">
  </head>
  <body>
    <div class = "app">
      <div class = "list">
      </div>
    </div>
```

```
    <script src = "myjs.js"></script>
  </body>
</html>
```

Полученный результат должен полностью совпасть с результатом, представленным на рис. 5.2.

5.8. Библиотеки JavaScript

С развитием WWW все большее значение придавалось простоте и скорости разработки при одновременном увеличении требований к качеству получаемого результата. Web-ресурсы должны были становиться все более привлекательными, с бóльшим количеством визуальных эффектов, с удобными возможностями обратной связи, но при этом быстро загружаться. Кроме того, в разных браузерах была по-разному реализована объектная модель документа (DOM), что обусловило необходимость затрачивать дополнительные усилия на реализацию корректного отображения страниц в различных браузерах. В результате было создано немало примеров решения таких задач, которые можно использовать в качестве готовых функций и шаблонов.

Библиотека JavaScript — это сборник готовых классов или функций пользовательского интерфейса, встраиваемых в web-приложения. Они не только содержат готовые решения типовых задач, встречающихся при создании динамических web-страниц, но и позволяют упростить взаимодействие JavaScript с другими языками, такими как CSS, PHP, Ruby и т. д.

Одной из наиболее популярных библиотек JavaScript является JQuery, фокусирующаяся на взаимодействии JavaScript, HTML и CSS.

Подключение JQuery. Библиотека подключается с использованием тега

```
<script src = "https://code.jquery.com/jquery-2.2.4.min.js"></script>
```

В этом случае нужные компоненты библиотеки в нужный момент подгружаются по CDN on-line.

Основы использования JQuery. Синтаксис оператора JQuery можно представить следующим образом:

```
$('селектор').действие('свойство действия)
```

Здесь:

— ***селектор*** — элемент или элементы, над которыми выполняется действие;

— ***действие*** — что именно мы будем делать с выбранными элементами (можно добавить какой-либо эффект, CSS-стиль, изменить HTML-код, указать какие-либо события);

— ***свойства действия*** — дополнительные атрибуты, связанные с данным действием.

Приведем несколько примеров использования JQuery.

Изменение стиля элементов страницы. Пусть необходимо изменить цвет заголовка страницы.

```
<!DOCTYPE html>
<html>
  <head>
    //Подключение библиотеки JQuery
    <script src = "https://code.jquery.com/jquery-2.2.4.min.js">
    </script>
    <meta charset = "UTF-8"/>
  </head>
  <body>
    <h1>Заголовок 1</h1>
    <p>первый абзац</p>
    <h2>Заголовок 2</h2>
    <p> Второй абзац  </p>
    <script>
      // в этом скрипте описывается JQuery-функция,
      // которая умеет менять стиль элементов
      $( function(){
        $("h1").css("color", "green");
      });
    </script>
  </body>
</html>
```

То, что функция начинается с $, означает, что будет использоваться JQuery. Вызов метода в оболочке этой функции означает, что он выполнится после наступления события загрузки страницы. Этот код найдет на странице все заголовки h1 и изменит их цвет на зеленый.

Обработка событий web-страницы. JQuery работает практически со всеми событиями в JavaScript, которые приводились выше. Аргумент this указывает, что действие будет произведено над текущим объектом.

Пример. При щелчке мыши по элементу абзаца цвет его шрифта будет изменен на красный.

```
$(function() {
  $("p").click(function() {
    $(this).css({
      "color": "red",
    });
  });
});
```

Аналогично можно организовать любую другую реакцию на любое событие любого элемента.

Интересные эффекты можно получить при анимации, вызванной некоторыми событиями.

Пример. Абзац после щелчка на нем мышью будет исчезать.

```
$(function() {
  $("p"). click(function() {
    $(this).hide();
  });
});
```

И, в заключение пример выпадающего меню, которое работает также как меню из п. 5.6, но не на «чистом» JavaScript, а с использованием библиотеки JQuery.

```
<!DOCTYPE HTML>
<html>
  <head>
    <script src = "https://code.jquery.com/jquery-2.2.4.min.js">
    </script>
    <meta charset = "UTF-8"/>
  </head>
  <body>
    // Структура меню абсолютно аналогичная
    // приведенной в п. 5.6
    <ul class = "menu" >
      <li>
        <p>Новости</p>
        <ul class = "submenu">
          <li >Новости культуры</li>
          <li >Новости спорта</li>
        </ul>
      </li>
      <li>
        <p>Наши продукты</p>
        <ul class = "submenu">
```

```
          <li >Продукт 1</li>
          <li >Продукт 2</li>
        </ul>
      </li>
    </ul>
    <script>
      // Функция доступна после загрузки документа
      $(document).ready(function () {
      /* Здесь используется так называемый псевдокласс hover,
которые описывает поведение при наведении/отведении мыши на эле-
мент */
      $('.menu li').hover(
      /* Срабатывает одна из функций которая раскрывает или закры-
вает все элементы ul от текущего (this). В скобках -- длительность
этой анимации в миллисекундах */
        function() {
          $('ul', this).slideDown(110);
        },
        function() {
          $('ul', this).slideUp(110);
        });
      });
    </script>
  </body>
</html>
```

Теперь даже без указания дополнительных стилей результат будет таким же, как на чистом JavaScript.

5.9. JS-фреймворки

Каждый год появляются все новые фреймворки, которые позволяют оптимизировать процесс «front end»-разработки сайтов. Среди них можно назвать Vue, Angular, React и др. В частности, один из наиболее популярных в настоящее время фреймворк React использует объектно-ориентированную методологию программирования и позволяет многократно использовать компоненты, которые объединяют любые объекты страницы. Преимущество состоит в том, что таким образом можно удобнее управлять целыми компонентами (наборами блоков) при создании интерфейса web-приложения.

Далее на втором примере из пункта 5.7 рассмотрим принципы применения React для вывода однотипных элементов на web-странице. В результате мы должны получить страницу, содержащую три однотипных компонента-карточки, каждая

из которых содержит три блока — картинка, подпись и пояснение. Компонент включает название, изображение и некоторое описание. Это классическая задача для React — управление компонентом, который содержит некоторые дочерние блоки или подкомпоненты.

С официального сайта React можно получить ссылки на внешние библиотеки, которые позволяют использовать готовые решения, реализующие описанные выше принципы компонентного подхода в React.

Итак, оставив неизменным файл стилей, внесем следующие изменения в файлы index.html и myjs.js.

Файл index.html:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8"/>
    <title>Демонстрация React</title>
    <link rel = "stylesheet" href = "styles.css">
    <!-- Добавление внешних ссылок на файлы из библиотеки React -->
    <script src = "https://unpkg.com/react@16/umd/react.
development.js"></script>
    <script src = "https://unpkg.com/react-dom@16/umd/react-dom.
development.js"></script>
    <script src = "https://unpkg.com/babel-standalone@6.15.0/
babel.min.js"></script>
  </head>
  <body>
    <!-- Добавление области для описания содержимого страницы -->
    <div id = "root"></div>
    <!-- Подключение внешнего собственного скрипта -->
    <!--Важно указать тип файла "text/babel”, чтобы новые возмож-
ности языка могли быть интерпретированы более старыми версиями,
которые могут использоваться в вашем браузере -- >
    <script type = "text/babel" src = "myjs.js"></script>
  </body>
</html>
```

Файл myjs.js:

```
/* Описываем функцию, которая создает шаблон для вывода на стра-
ницу одной карточки с экземпляром галереи. Коллекция значений,
которые могут подставляться в этот шаблон всегда описывается
в React как props */
function Comp(props) {
return (
<div className = "card">
```

```
  <div className = "card-img">
    <img
      src = {props.comp.pict}
      />
    <h3>{props.comp.name}</h3>
  </div>
  <p>{props.comp.note}</p>
</div>
)
}

/* Описываем класс компонента App (приложение), наследуемого
от компонента React, в котором имеется его состояние (state) -
в данном случае набор всех возможных значений, и рендеринг
(render) - отрисовка на странице шаблона со связанным с ним состо-
янием */
class App extends React.Component {
  state = {comps:
  [
    {pict:'img/330px-Apple-II.jpg',
      name:'Apple-II',
      note:'год выпуска - 1977'},
    {pict:'img/330px-IBM_5150_PC.jpg',
      name:'IBM-5150',
      note:'год выпуска - 1982'},
    {pict:'img/330px-Macintosh_128k_transparency.png',
      name:'Macintosh_128',
      note:'год выпуска - 1984'}
  ]
}

render()
{
  return(
    <div className = "app">
      <div className = "list">
  {
  /* Здесь используется метод map - отображение каждого из элемен-
тов в списке comps на его место в шаблоне Comp*/
      this.state.comps.map(comp = >
      {
        return(
          <Comp comp = {comp}/>
        )
      })
    }
    </div>
  </div>
  )
  }
}
```

```
// Вывод отрисованных шаблонов в отведенное для них место на стра-
нице
ReactDOM.render(
  <App />
  , document.getElementById('root'))
```

Результат должен совпасть с представленным на рис. 5.2.

Нельзя считать React полноценным JS-фреймворком, так в нем реализуются методы, позволяющие более удобно представлять только уровень представления, *view* (подробнее об этом в теме 8 пособия). Однако его использование предоставляет разработчику возможность создавать более простой и понятный код на стыке HTML и JavaScript за счет того, что HTML-фрагменты с использованием специального синтаксиса можно включить прямо внутрь JS-функций. В примере выше вы видите пример применения такого синтаксиса — круглые скобки после оператора return.

Контрольные вопросы

1. На какой стороне в клиент-серверной технологии выполняются JavaScript?
2. Перечислите основные возможности языка JavaScript.
3. Как можно добавлять код JavaScript в web-документ?
4. Как задаются комментарии в тексте JavaScript?
5. Как объявляются переменные JavaScript?
6. Какие типы переменных JavaScript Вам известны?
7. Какие операторы сравнения имеются в JavaScript?
8. Какие виды модальных окон Вам известны? Опишите соответствующие им операторы.
9. Опишите оператор условия в JavaScript
10. Опишите операторы цикла в JavaScript
11. Как и для чего используются функции JavaScript?
12. Для чего используются обработчики событий в JavaScript? Приведите примеры.
13. Почему возникла необходимость в разработке дополнительных библиотек JavaScript?
14. Перечислите основные возможности библиотеки JQuery
15. Какие еще библиотеки JS вам известны? Самостоятельно изучите и проанализируйте их назначение и особенности.
16. Поясните различия между JS-библиотекой и JS-фреймворком.
17. Поясните сущность компонентного подхода, используемого фреймворком React.

Контрольные задания

1. Измените текст страницы так, чтобы скрипт печатал все четные числа от 2 до 20.

```
<html>
  <body>
    <script type = "text/javascript">
      let i = 0;
      for (i = 0;i< = 5;i++)
      {
        alert("Число i равно " + i);
      }
    </script>
  </body>
</html>
```

2. Добавьте на страницу функции и демонстрацию работы функций изменения содержимого абзаца (а не заголовка) при наведении на него мыши.

```
<html>
  <head>
    <script type = "text/javascript">
      function nameon()
      {
        document.getElementById('h2text').innerHTML = "ДОБРО
        ПОЖАЛОВАТЬ!";
      }
      function nameout()
      {
        document.getElementById('h2text').innerHTML = "Как
        дела сегодня?";
      }
    </script>
  </head>
  <body>
    <h2 id = "h2text" onmouseout = "nameout()" onmouseover =
"nameon()">
      Переместите мышь над этим текстом!
    </h2>
  </body>
</html>
```

3. По приведенному примеру (вычисления площади прямоугольного треугольника) составьте скрипт, который рассчитывает площадь круга и длину окружности по введенному радиусу.

```
<html>
  <head>
    <title>ploshad</title>
```

```
    <script type = "text/javascript">
      function ploshad_tr()
      {
        a = document.forms["f1"].elements["k1"].value;
        b = document.forms["f1"].elements["k2"].value;
        document.getElementById('txt').innerHTML =
        "площадь"+a*b/2;
        alert(a*b);
      }
    </script>
  </head>
  <body>
    <form id = "f1" name = "f1">
      <input text = "text" id = "k1" name = "k1" />
      <input text = "text" id = "k2" name = "k2" />
      <input type = "button" value = "Отправить" onClick =
"ploshad_tr();" />
    </form>
    <p id = "txt">
  </body>
</html>
```

4. Выполните предыдущее задание с использованием возможностей JQuery.

5. Создайте минигалерею, содержащую 6 любых изображений и подписей к ним с использованием HTML, «чистого» JavaScript и React.

Тема 6
ОСНОВЫ СЕРВЕРНОЙ ОБРАБОТКИ WEB-ПРИЛОЖЕНИЙ

После изучения данного раздела студент должен:

знать

- основные возможности и область применения серверных языков программирования и языка PHP,
- правила оформления кода на языке PHP,
- типы переменных языка, особенности их применения и принципы преобразования,
- особенности операций с различными типами данных в языке PHP,
- особенности применения условных и циклических конструкций в языке PHP,
- особенности работы с массивами,
- правила описания и применения функций;

уметь

- составлять программы на языке PHP и использовать их совместно с кодом HTML,
- применять переменные различных типов, встроенные функции языка, условные и циклические конструкции, массивы разных типов,
- организовывать вывод данных на языке PHP с использованием структурирования на языке HTML;

владеть

- навыками настройки среды разработки серверных web-приложений с использованием одного из локальных web-серверов и инструментов разработчика.

6.1. Серверное программирование. Назначение и возможности PHP

В предыдущих разделах данного учебного пособия мы разобрали понятие клиент-серверного взаимодействия и выяснили, какие технологии, технические и языковые средства использу-

ются при работе web-приложений на стороне клиента. Далее речь пойдет о технологиях, которые используются на сервере. Считается, что для разработчика серверное программирование проще, чем клиентское, так как обработка запросов на сервере требует, как правило, довольно стандартных, однотипных операций обращения к базе данных и формирования ответов.

Важнейшей частью серверных технологий при организации web-приложений является web-сервер. **Web-сервером** называют как программное обеспечение, выполняющее функции обработки HTTP-запросов, поступающих от браузеров, так и непосредственно компьютер, на котором это программное обеспечение работает.

В зависимости от функций приложения может понадобиться либо статический, либо динамический web-сервер.

Статический web-сервер способен высылать ответы на HTTP-запросы в виде готовых статических страниц, которые размещены на нем.

Динамический web-сервер, в дополнение к возможностям статического, содержит программное обеспечение, которое способно отправлять запросы к базе данных (хранящейся на отдельном сервере баз данных или встроенной в web-сервер), обрабатывать эти запросы при необходимости и отправлять браузеру полученную информацию в составе HTML-страницы. Например, при работе с некоторым электронным магазином пользователь в ответ на свой запрос о некотором товаре получит не готовую, хранящуюся на сервере, страницу об этом товаре, а шаблон страницы товара, в которую встроены из базы данных (БД) сведения именно об этом товаре.

Серверное программное обеспечение (сценарии), которое способно выполнять такие функции, пишется на одном из ***серверных языков web-программирования***. К этим языкам относят: PHP, Python, Rubi, Node.JS, Java и др.

Наиболее распространенные программы-web-серверы, которые способны «понимать» программы, написанные на этих языках — это Apache, Microsoft Internet Information Server, Netscape и iPlanet.

Для удобного создания, изменения и обработки динамически изменяющихся данных (списков пользователей, товаров, блогов и т. д.) используются *системы управления базами данных* (СУБД).

Далее мы будем рассматривать PHP — широко используемый язык серверных сценариев общего назначения с откры-

тым исходным кодом. Аббревиатура PHP означает «*Hypertext Preprocessor*» (Препроцессор Гипертекста). Синтаксис языка очень похож на синтаксис C, поэтому язык довольно прост в изучении. PHP реализован для большинства операционных систем, включая Linux, многие модификации Unix (такие как HP-UX, Solaris и OpenBSD), Microsoft Windows, Mac OS, RISC OS, и многих других. Также в PHP включена поддержка большинства современных web-серверов. Еще одним значительным преимуществом PHP является поддержка широкого круга СУБД.

Схема работы web-сервера Apache со скриптом на PHP приведена на рис. 6.1.

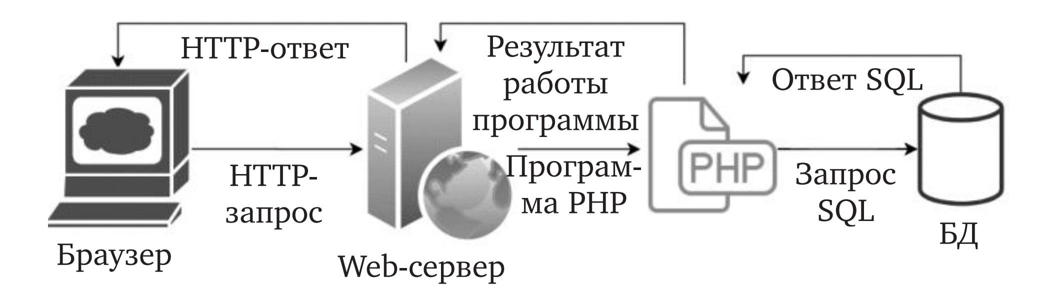


Рис. 6.1. **Серверная обработка web-приложения**

Пользователь через браузер отправляет запрос к web-серверу по протоколу HTTP. Web-сервер определяет, к какому типу файла было совершено обращение, и, если это файл с расширением .php, то запрос передается на обработку интерпретатору языка PHP, скрипт обрабатывается, выполняется, при необходимости запрашивает и получает некоторые сведения из базы данных и возвращает ответ web-серверу. Web-сервер передает сформированный результат пользователю. Этот результат, как правило, представляет собой динамически сгенерированную HTML-страницу.

Скрипты на PHP могут встраиваться в код HTML-страницы (страница при этом должна получить расширение. php) или записываться в отдельные скриптовые файлы, которые содержат только php-функции.

6.2. Основы синтаксиса и типы переменных PHP

Код заключается в специальные теги PHP ***<?php*** и **?>.**
Каждый оператор заканчивается точкой с запятой.

Оператор **echo**, осуществляет вывод информации в браузер.

```
<?PHP
  echo 'Hello!';
?>
```

Можно также для вывода информации на страницу использовать оператор print:

```
<?PHP
  print 'Hello!';
?>
```

В PHP используют 3 типа комментариев.

```
# Это однострочный комментарий
// Это тоже однострочный комментарий
/* А это
длинный многострочный
комментарий
*/
```

Как и в других языках программирования, переменной называется именованная область памяти, в которой содержатся данные. Имена переменных в PHP начинаются со знака доллара «$» и состоят из цифр, букв и знака подчеркивания.

Примеры различных переменных в PHP:

```
$word
$my_word
$word24
$ВЫСОТА
```

Есть набор правил, которых нужно придерживаться при назначении имен переменных:

— в имя переменной буквы кириллицы могут входить как полностью, так и частично;

— после знака доллара в имени переменной может идти только символ подчеркивания или буква;

— переменные $word и $worD будут интерпретироваться как 2 разные переменные, то есть имена переменных в PHP — чувствительны к регистру;

— имена переменных в PHP не должны совпадать с ключевыми словами этого языка. Список ключевых слов можно увидеть в официальной документации PHP;

— длина имени переменной в PHP не ограничена.

В PHP существуют различные **типы данных**:
— вещественные числа (float, real);
— целые числа (integer);
— строки (string);
— логические величины (boolean);
— массивы (array);
— обьекты (object) и некоторые другие.

PHP сам способен определять тип переменной, который он «узнает» исходя из данных, которые хранятся в переменной. Пример:

```
<?php
  /* Сначала объявляем переменную, с одновременной инициализацией.
Таким образом косвенно определятся ее тип - integer*/
  $number = 0;
  // Осуществляем операцию сложения
  $result = $number + 20;
  print $result;
?>
```

PHP не требует явного указания типа при определении переменной; тип переменной определяется по контексту, в котором она используется. Это значит, что если вы присвоите значение типа string переменной $var, то $var изменит тип на string. Если вы затем присвоите $var значение типа integer, она станет целым числом (integer).

Примером автоматического преобразования типа является оператор умножения (*). Если какой-либо из операндов является типом float, то все операнды интерпретируются как float, и результатом также будет значение типа float.

Однако можно использовать и специальные функции приведения типов. Имя требуемого типа записывается в круглых скобках перед приводимой переменной.

```
<?php
  $first = 10; // $first - это целое число
  $second = (float)$first;//теперь $second - это действительный тип
?>
```

Допускаются следующие функции приведения типов:
— (int), (integer) — приведение к integer;
— (bool), (boolean) — приведение к boolean;
— (float), (double), (real) — приведение к float;

— (string) — приведение к string;
— (array) — приведение к array.

Если взять любое число и заключить его в кавычки, то данная последовательность будет воспринята интерпретатором PHP как строка, а не как число. Важно понимать ***разницу между использованием двойных и одинарных кавычек в PHP***, так как неправильный выбор кавычек в некоторых случаях может привести к неработоспособности web-приложения.

Внутри двойных кавычек происходит интерпретация переменных, в результате чего на выходе мы получаем их значения.

Внутри одинарных кавычек переменные не интерпретируются, то есть строка пишется как есть.

```
$name = "Саша";
$sentence1 = "Его зовут $name";
$sentence2 = 'Его зовут $name';
echo $sentence1;
echo $sentence2;
```

В первом случае мы получаем ответ «Его зовут Саша», а во втором — «Его зовут $name».

Мы не будем специально останавливаться на правилах использования арифметических операторов в PHP. Они практически идентичны тем, которые были описаны для JavaScript в предыдущем разделе пособия. Но стоит оговориться по поводу двух операций. Операция деления (/) возвращает число с плавающей точкой, кроме случая, когда оба значения являются целыми числами, которые делятся нацело — в этом случае возвращается целое значение. Для целочисленного деления дробных операндов нужно использовать ***intdiv()***. При делении по модулю операнды преобразуются в целые числа (удалением дробной части) до начала операции. Для деления по модулю чисел с плавающей точкой нужно использовать ***fmod()***.

Со встроенными математическими функциями, которые используются в PHP Вы можете ознакомиться по ссылке http://PHP.net/manual/ru/ref.math.PHP.

6.3. Условные и циклические операторы в PHP

В языке PHP есть несколько способов записи конструкции **if**. В самом простом способе if проверяет ложность или истинность определенного условия и в зависимости от результата

проверки выполняет или не выполняет группу выражений, размещенных в фигурных скобках.

В данном случае конструкция if будет иметь следующий формат записи:

```
<?php
  if(какое-либо условие) {
    блок выражений
  }
?>
```

Пример:

```
<?php
  $b = 5;
  if($b < 10) {
     echo $b . "меньше десяти";
     }
?>
```

В этом примере переменной $b присваивается значение 5. Потом осуществляется сравнение 5 < 10. Как мы видим, оно истинно, поэтому заключенный в фигурные скобки код будет выполняться. В результате на экран будет выведена фраза: 5 меньше десяти.

Приведенный выше способ записи может быть усложнен с помощью команд elseif и else. Синтаксис расширенного формата следующий:

```
<? php
  if(какое-либо условие) {
    Блок выражений 1
  }
  elseif (другое условие) {
    Альтернативный блок выражений 2
  }
  else {
    Альтернативный блок выражений 3
  }
?>
```

Конструкция if, как и любая другая условная конструкция, использует условные выражения, например $chislo > = 0. Это выражение истинно, если переменная $chislo больше или равняется нулю. Операнды можно сопоставлять не только на ра-

венство, но и на неравенство. Для этого используют оператор ! (not).

В PHP также можно объединять условные выражения в пределах одной конструкции. На практике довольно часто нужно проверить операнды на несколько соответствий. Например, узнать, входит ли число в диапазон чисел от 20 до 100. Для таких целей используются операторы OR, AND. С их помощью условные выражения можно комбинировать.

Оператор OR возвращает значение TRUE, если хотя бы одно выражение является истинным.

Оператор AND возвращает значение TRUE, если истинными одновременно будут оба выражения — слева и справа.

Пример с оператором OR:

```
<?php
  $login = "Вася";
  //Теперь проверим, совпадает ли $login с одним из заданных логинов
  if($login == "Вася" OR $login == "Маша") {
    echo 'Логин правильный';
  }
  else {
    echo 'Вы ввели ошибочный логин';
}
?>
```

В данном примере оператор OR используется для проверки строки (логина) на совпадение с одной из 2 строк. Выражение вернет TRUE, если будет хотя бы одно совпадение.

Пример с оператором AND:

```
<?php
  $b = 15;
  // Проверяем, входит ли переменная $b в диапазон 10 < $b < 100
  if ($b > 10 AND $b < 100) {
    echo'$b входит в диапазон';
  }
  else {
    echo'$b не входит в диапазон';
  }
?>
```

В этом примере выражение вернет значение TRUE только в том случае, если значение переменной $b одновременно будет больше десяти и меньше 100.

Для управления порядком сравнения условных выражений, как вы возможно заметили в примере, их нужно заключать в круглые скобки.

Следующая запись используется для проверки существования той или иной переменной:

```
<?php
  if(!$b){
     echo 'Переменной $b не существует';
  }
?>
```

Но следует заметить, что для проверки существования переменных лучше использовать встроенную функцию PHP ***isset()***.

```
<?php
  if(isset($b)) {
    echo 'Такая переменная существует';
  }
  else {
    echo 'такой переменной не существует';
  }
?>
```

Функция ***empty()*** используется для того, чтобы узнать, пусто ли значение переменной. Иногда переменная в PHP скрипте может быть установлена и функция isset() вернет TRUE, но значение переменной будет пустым. Функция empty() используется для дополнительной проверки. Формат записи:

```
<?php
  $b = "";
  if(!empty($b)) {
    echo 'Значение переменной $b не является пустым';
  }
?>
```

Для повторения блока инструкций несколько раз в PHP, как и в других языках программирования, используются операторы цикла. В языке программирования PHP существует несколько способов записи циклических конструкций:

1) while;
2) for;
3) do… while;
4) foreach.

Циклическую конструкцию ***while*** используют в случаях, когда количество повторений неизвестно заранее. Повторение будет выполняться до тех пор, пока условие цикла равняется true.

Синтаксис цикла while:

```
while (условие продолжения) {
  //блок команд
}
```

Пример цикла while:

```
<?php
  // Присваиваем переменной начальное значение
  $begin = 0;
  // Устанавливаем предельное значение
  $end = 9;
  // Начинаем цикл с условием
  while ($begin < = $end) {
    // Определяем номер итерации
    $iteration = $begin + 1;
    echo "$iteration –я итерация<br />";
    //Обратите внимание!!! Здесь мы увеличиваем значение $begin!
    $begin++;
  }
?>
```

Данный скрипт выводит номера повторений цикла. Отсчет начинается со значения $begin, которое устанавливается в перед циклом. Потом переменная $end определяет номер последней итерации, на которой цикл останавливается.

В отличие от цикла while, цикл ***for*** в основном используют для выражений с заранее известным количеством повторений. Но for отличается тем, что условие меняется в самой конструкции, а не снаружи или внутри блока скрипта.

Синтаксис цикла for:

```
for (инициализация; условие; приращение)
{
  //блок команд
}
```

Пример использования цикла for:

```
<?php
  for ($begin = 0, $end = 9; $begin < = $end; $begin++) {
    // Присваиваем номер итерации
    $iteration = $begin+1;
```

```
    echo "$iteration –я итерация<br />";
  }
?>
```

Во время первого прохода данной циклической конструкции инициализируются переменные $begin и $end: они получают значения 0 и 9. Потом интерпретатор проверяет истинность условия цикла. Конечно, 0 < 9, поэтому блок кода внутри цикла for выполняется, а $begin увеличивается на 1. После вывода echo условие цикла проверяется снова. В этот раз переменная $begin = 1, что меньше 9, поэтому цикл продолжает выполняться. Когда значение $begin = 10, условие $begin < = $end оказывается ложным (false) и цикл заканчивается.

Циклическая конструкция ***do…while*** очень похожа на цикл while. Отличается тем, что условие проверяется не в начале, а в конце выполнения каждого прохода тела цикла. Таким образом, всегда существует одна обязательная итерация.

Пример:

```
<?php
  $begin = 0;
  $end = 10;
  do {
    echo $begin;
  }
  while($begin> = $end);
?>
```

Так как выражение 0 > = 10 является ложным, условие цикла должно вернуть значение false. Однако, как было замечено раньше, одна итерация данного цикла выполняется всегда. Поэтому на экране в результате выполнения скрипта появится значение переменной $begin, а именно, 0.

6.4. Массивы в PHP

Массив — это организованная совокупность ячеек памяти, хранящих данные под одним именем. Массив может одновременно содержать несколько элементов (значений). Элементы массивов идентифицируются по их индексам (ключам). Любой массив состоит из одной или нескольких пар «ключ — значение».

Простым примером массива может быть обычное слово. Все строковые данные в PHP можно представить в виде массива.

Например, слово «программа» — это массив, который включает девять элементов. Для обращения к элементу массива нужно указать его индекс. В языке программирования PHP, как и во многих других языках, индексация начинается с 0. Индекс первого элемента массива (в нашем случае это буква «п») будет равняться нулю.

Пример:

```
<?php
  $word = "программа";
  echo $word[0];
  // на экран будет выведена буква "п"
  // в примере ключ элемента – 0, значение элемента – "п".
?>
```

Массив можно создать с помощью конструкции array(). Эта конструкция позволяет создавать пустой массив:

```
<?php
  $moy_massiv = array();
?>
```

или массив с некоторыми элементами:

```
<?php
  $moy_massiv = array("10", "20");
?>
```

До сих пор мы говорили про одномерные массивы. Массив в таком случае имеет очень простую структуру: «ключ => значение».

Но часто приходится иметь дело с более сложными структурами данных. Хорошим примером многомерных данных может быть список зарегистрированных пользователей сайта. В массиве данных хранится информация не только о логине и пароле, но и другая частная информация о пользователях. Многомерные массивы используются для описания именно таких структур.

У многомерного массива каждый элемент имеет не менее двух индексов. Это достигается за счет того, что в PHP элементом массива может выступать любой тип данных, и другой массив в том числе. Создаются своего рода «массивы массивов». Для примера рассмотрим многомерный массив пользователей сайта:

```
<?php
  $polzovateli = array (
    0 => array (
      "login" => "Admin",
      "paswd" => "admin',
      "email" => "admin@mail.ru",
      "profession" => "PHP программист"
    ),
    1 => array (
      "login" => "Sasha",
      "paswd" => "Sasha1",
      "email" => "alex@mail.ru",
      "profession" => "инженер"
    ),
    2 => array (
      "login" => "Petr",
      "paswd" => "Petr1",
      "email" => "petr@mail.ru",
      "profession" => "JS программист"
    )
  );
?>
```

Для доступа к элементам многомерного массива индексы записывают сразу после предыдущих в квадратных скобках. Если добавить к вышеприведенному коду такую строку:

```
echo $polzovateli [1] ["profession"];
```

то мы получим профессию пользователя — инженер.

Многомерные массивы делятся на:

1) ***индексные*** (ключами массивов выступают целочисленные значения);

2) ***ассоциативные*** (ключами являются исключительно строковые данные);

3) ***смешанные*** (логично, что здесь ключами могут быть как целые числа, так и строковые величины).

Как видите, в данном примере рассматривался смешанный многомерный массив.

При работе с массивами в языке программирования PHP множество задач решается с помощью перебора их элементов. Для этого используют рассмотренные в прошлых уроках циклические конструкции (for, while…) или конструкцию ***foreach***, которая была специально создана для работы с массивами.

Сравним программы для перебора элементов некоторого массива, созданных с использованием традиционного цикла (оператор while) и цикла, с использованием оператора foreach.

```
<?php
  $goroda = array ("Москва", "Курск", "Вологда");
  $index = 0;
  $elements = count ($goroda);
  while ($index < $elements) {
    echo $index+1 . "." . $goroda[$index] . "<br>";
    $index++;
  }
?>
```

В результате работы этого скрипта мы увидим на экране следующую информацию:

```
1. Курск.
2. Вологда.
3. Москва.
```

В начале создается массив $goroda с 3 элементами:
$goroda [0] = «Курск»,
$goroda [1] = «Вологда»,
$goroda [2] = «Москва».

Потом инициализируются 2 переменные: $index и $elements. Первая выступает ключом (индексом) для последующего обращения внутри цикла к элементам массива. Эта переменная получает значение 0, так как индексация массивов в PHP начинается с нуля.

Переменная $elements получает значение, которое вернула встроенная функция count(). Встроенная функция count() возвращает количество элементов массива, имя которого задается в аргументе. В примере 3 элемента, поэтому значение $elements будет равняться трем.

Итерация в цикле согласно условию выполняется до тех пор, пока индекс элементов не будет равняться максимальному количеству элементов в перебираемом массиве (не забываем, что переменная $index равна нулю, а не единице, поэтому в операторе сравнения стоит знак <, а не <=). На экран при помощи команды echo выводится результат операции конкатенации (сцепления строк). В конце выполняется инкремент (увеличение переменной $index на единицу). Не следует забывать об этом шаге, так как он позволяет работать с последующими элементами массива и обеспечивает прекращение работы цикла, когда заданное условие выполнено.

Такой же результат можно получить с использованием оператора foreach:

```
<?php
  $goroda = array ("Москва", "Курск", "Вологда");
  $index=1;
  foreach ($goroda as $value){
    echo $index . "." . $value . "<br>";
    $index++;
  }
?>
```

Здесь оператор foreach указывает, что нужно выполнить тело цикла для каждого значения ($value), входящего в массив.

Еще более эфективно использование этого оператора цикла для ассоциативных массивов. В данном случае мы можем сразу обращаться ко всем входящим в ассоциативный массив парам ключ ($key) — значение ($value):

```
<?php
  $goroda = array (1=>"Москва", 2=>"Курск", 3=>"Вологда");

  foreach ($goroda as $key=>$value){
    echo $key . "." . $value . "<br>";
  }
?>
```

Таким образом, язык PHP позволяет организовать достаточно сложные структуры данных и эффективно их обрабатывать.

6.5. Функции PHP

Функция — это некоторый набор команд, как правило, реализующих какую-либо законченную задачу, который хранится под некоторым именем. Основная задача создания функций — это организация программы таким образом, чтобы правильно работающий код можно было многократно использовать, не изменяя и не создавая снова. То есть та часть программы, которая может понадобиться снова, записывается в виде функции и по мере необходимости просто вызывается, через указание ее имени.

Существуют два типа функций: встроенные и пользовательские. Встроенные функции — это функции, которые за нас уже написали создатели языка программирования, и мы можем просто их использовать. В PHP существуют тысячи готовых функций, которые вы можете найти в различных источниках[1].

[1] См. напр.: https://1st-network.ru/prog/in-functions

Далее мы будем рассматривать только пользовательские функции.

Общая схема описания функции PHP такова:

```
<?php
  function <имя функции>(<аргумент функции>) {
    <тело функции>
    return <результат выполнения функции>;
  }
```

Так, например, пусть нам необходимо выяснить, является ли пользователь совершеннолетним:

```
<?php
  function is_year($year) {
    if ($year >16) {
      return true;
    }
    else {
      return false;
    }
  }
```

Мы описали функцию с именем is_year, которая принимает в качестве аргумента значение $year (например, возраст пользователя) и возвращает логическое значение true, если возраст больше 16 лет, и false — в противном случае.

В нужном месте программы эта функция может быть вызвана:

```
<?php
  $year = 15;

  if (is_year($year)) {
    print("Вы совершеннолетний");
  }
  else {
    print ("Вы-несовершеннолетний");
  }
?>
```

Таким образом, нами были рассмотрены наиболее общие принципы и методы применения языка PHP. Далее будут рассмотрены возможности работы на PHP с базами данных.

Контрольные вопросы

1. Назовите основные возможности языка PHP. Для решения каких задач он обычно применяется?

2. Как оформляется код на PHP? Какое расширение должен иметь файл, содержащий такой код?

3. Какого типа переменные используются в PHP? Как описывается тип переменных?

4. Каковы особенности использования кавычек в PHP?

5. Каковы основные арифметические операции в PHP?

6. Опишите синтаксис условного оператора в PHP. Приведите примеры.

7. Какие основные логические операторы, используемые в PHP вам известны?

8. В чем состоит назначение и какой синтаксис оператора while? Приведите примеры.

9. В чем состоит назначение и какой синтаксис оператора for? Приведите примеры.

10. В чем состоит назначение и какой синтаксис оператора do… while? Приведите примеры.

11. В чем состоит назначение массивов в языке PHP. Какие типы массивов в нем используются? Приведите примеры.

12. В чем состоит назначение и каков принцип использования оператора foreach?

13.Для чего используются функции языка PHP? Приведите пример.

Контрольные задания

Используя инструкцию по установке web-сервера, приведенную в практикуме данного учебного пособия, настройте среду выполнения программ на языке PHP и решите следующие задачи.

1. Создайте переменные $c = 15 и $d = 2. Просуммируйте их, а результат присвойте переменной $result. Выведите на экран значение переменной $result.

2. Создайте переменную $word и присвойте ей значение ‘hello’. Обращаясь к отдельным символам этой строки выведите на экран символ ‘h’, символ ‘e’, символ ‘o’.

3. Создайте ассоциативный массив $arr, содержащий сведения о пяти сотрудниках и их зарплатах. Выведите на экран зарплату двух сотрудникова.

4. Выведите в HTML-таблицу все значения ассоциативного массива, созданного в предыдущем задании.

5. Выведите каждый из элементов некоторого массива шрифтом разного стиля.

6. Выведите меньшее из трех введенных в переменные чисел.

7. Если целое число a делится нацело на целое число b, выведите результат, если нет — выведите сообщение об этом.

8. Известны две скорости: одна в километрах в час, другая в метрах в секунду. Выведите ту, которая больше.

9. Дано двузначное число. Определите, какая из двух его цифр больше и выведите сообщение об этом.

10. Пользователь вводит логин и пароль. Если они совпадают с некоторыми известными программе значениями, выведите приветствие, иначе — предложите ввести логин и пароль заново.

Тема 7
ПРИМЕНЕНИЕ PHP ДЛЯ РАБОТЫ С БАЗОЙ ДАННЫХ

После изучения данного раздела студент должен:

знать

- место и роль баз данных и СУБД в разработке web-приложений,
- преимущества использования баз данных при создании динамических приложений,
- наиболее используемые при разработке web-приложений СУБД,
- основы языка SQL,
- возможности и принципы обмена данными между приложением и базой данных,
- возможности языка PHP по обращению к базам данных и представлению полученной информации на web-странице;

уметь

- создавать простую реляционную базу данных в среде одной из СУБД,
- писать несложные запросы к базе данных на языке SQL,
- описывать команды выполнения запросов вставки, удаления и обновления к базе данных средствами языка PHP,
- организовывать вывод данных запроса на web-страницу;

владеть

- навыками использования СУБД для создания структуры, заполнения и просмотра баз данных,
- подключения к базе данных на языке PHP,
- отладки программ и использования механизмов поиска ошибок программного выполнения запросов к базе данных на языке PHP.

7.1. Зачем использовать базы данных в web-разработке

Работа с базами данных — это одна из важнейших составляющих программирования сайтов динамического типа. Будь

то формирование страниц «на лету» или же реагирование на действия посетителей сайтов — почти все эти задачи требуют взаимодействия с базами данных.

Базы данных (БД) для сайтов используются с целью хранения различной информации и, упрощенно, представляют собой некоторый набор взаимосвязанных таблиц. Именно в базах данных хранится на сервере требуемая для работы сайта информация, например, информация о клиентах, каталог товаров, статистические данные и т. д.

Управление БД как правило, осуществляется при помощи клиент-серверных СУБД, таких как Oracle, MS SQL Server, PostgreSQL, MySQL и др. Клиент-серверные СУБД обрабатывают запросы централизованно, к их достоинствам относят обеспечение высокой надежности баз данных, их доступности и безопасности.

Для построения запросов к базам данных широко применяется SQL (*Structured Query Language*) — «язык структурированных запросов». С помощью SQL может осуществляться добавление, удаление, редактирование записей в таблицах БД, выборка данных в соответствии с различными условиями, сортировка данных и многое другое.

Язык PHP для создания web-приложений обычно применяют в связке с СУБД MySQL, хотя на нем можно писать программы, которые работают с базами данных, управляемыми и другими СУБД.

В практической части данного курса мы будем реализовывать клиент-серверное взаимодействие с применением пакета разработчика OpenServer, который позволяет создавать и отлаживать свои web-приложения локально. Этот пакет поддерживает множество web-серверов, среди которых Apache, Nginx, почтовых и DNS-серверов, несколько различных реляционных и нереляционных СУБД, PHP самых последних версий.

7.2. Основы СУБД MySQL

MySQL — это система управления ***реляционными*** базами данных. Это означает, что данные в таких базах хранятся в отдельных таблицах, которые связываются между собой. Этот способ хранения данных обеспечивает с одной стороны минимальную избыточность данных, а, с другой, эффективное выполнение любых типов запросов на поиск информации.

База данных состоит из таблиц, а каждая таблица состоит из строк (записей). Один сервер MySQL может поддерживать сразу несколько баз данных, доступ к которым может разграничиваться логином и паролем. Зная эти логин и пароль, можно работать с конкретной базой данных. Например, можно создать или удалить в ней таблицу, добавить записи и т. д. Обычно имя-идентификатор и пароль назначаются хостинг-провайдерами, которые и обеспечивают поддержку MySQL для своих пользователей.

Существует несколько способов создания новой базы данных MySQL:

— через запросы на языке SQL

— через интерфейс, предоставляемый СУБД. В нашем случае это средство называется PHPMyAdmin.

Первый способ требует достаточно глубоких знаний в теории баз данных, поэтому в рамках данного курса ограничимся вторым способом.

Для создания базы данных в среде PHPMyAdmin необходимо выполнить следующие шаги:

1) создать пустую базу данных;
2) создать пустую таблицу в этой базе данных;
3) описать структуру этой таблицы (набор столбцов);
4) ввести конкретные данные в эту таблицу;
5) если база данных должна состоять из нескольких таблиц, то необходимо повторить пункты 2—4 для каждой таблицы.

Описание структуры (набора столбцов) требует задания следующей информации: имя столбца, тип столбца, информация о ключах, значение по умолчанию (при необходимости).

При определении имени столбца необходимо помнить лишь то, что имя не должно содержать пробелов и должно быть содержательно понятным, например cena_tovara (Цена товара)

Наиболее часто встречающиеся типы данных приведены в табл. 7.1.

Таблица 7.1.

Типы данных СУБД MySQL

Тип	Название в MySQL	Описание, диапазон возможных значений
Целый	INT	–2 147 483 648 ÷ 2 147 483 647
Вещественный	FLOAT	Небольшая точность
	DECIMAL	Дробное число, хранящееся в виде строки

Окончание табл. 7.1

Тип	Название в MySQL	Описание, диапазон возможных значений
Символьный	CHAR	Строка длиной до 255 символов
	VARCHAR	Строка длиной до 255 символов (обрезается до реального размера +1 символ)
Строка	TEXT	Текст с максимальной длиной 65 535 символов (64 Кб)
Дата и время	DATE	Дата в формате ГГГ-ММ-ДД
	TIME	Время в формате ЧЧ:ММ:СС
	DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС

Каждая таблица должна иметь ключевой столбец. Он содержит данные, которые однозначно определяют другие данные в этой строке таблицы. Примером ключевого столбца может быть библиотечный код книги или номер студенческого билета студента.

Приведем пример описания структуры таблицы для реализации базы данных в среде СУБД MySQL (табл. 7.2).

Таблица 7.2

Таблица knigi

Имя столбца	Тип столбца	Длина	Сведения о ключах	Пояснения
bibl_kod	VARCHAR	20	Первичный ключ	библиотечный код
name_knigi	VARCHAR	100	—	название
avtor	VARCHAR	100	—	автор
god_izd	INT	—	—	год издания
izdatelstvo	VARCHAR	100	—	издательство
kol_stranic	INT	—	—	количество страниц
cena	DECIMAL	—	—	цена книги
data_vidachi	DATE	—	—	дата выдачи книги читателю
annotacia	TEXT	—	—	краткое содержание

В реальных базах данных таблицу с такой структурой использовать нежелательно, она не соответствует многим правилам проектирования промышленных баз данных и используется здесь только в качестве иллюстрации описания всех типов полей таблиц MySQL.

После описания такой структуры таблицы PHPMyAdmin предложит пользователю пустую форму с окнами, в которые можно будет вводить строки таблицы с конкретными сведениями о книгах.

7.3. Некоторые возможности языка манипулирования данными SQL для работы с базами данных

В данном курсе мы изучаем технологии баз данных для того, чтобы уметь сохранять данные, введенные на сайте, в базе данных на сервере, а затем, при необходимости, выводить их на страницах сайта. Поэтому нам важно выяснить, как можно манипулировать данными из БД программно, из PHP скриптов.

На самом деле PHP только создает интерфейс для запросов к базе данных, а сами запросы пишутся на языке SQL. Структурированный язык запросов SQL позволяет производить различные операции с базами данных: создавать таблицы, помещать в них новые данные, обновлять, удалять из них данные, отбирать нужные сведения из таблиц и т. д. Далее рассмотрим основные из этих операторов, что позволит понять основные принципы работы с базой данных в web-программировании.

Примечание

Команды SQL не чувствительны к регистру, но традиционно они набираются прописными буквами.

Рассмотрим базу данных «Книги», состоящую из одной таблицы knigi, описанной в предыдущем разделе.

Добавление данных в таблицу MySQL. Для добавления записей используется оператор **INSERT**:

```
INSERT INTO Имя_таблицы [(Список полей)] VALUES (Список констант)
```

После выполнения оператора INSERT будет создана новая строка таблицы, в качестве значений полей будут исполь-

зованы соответствующие константы, указанные в списке VALUES.

Рассмотрим пример использование оператора INSERT.

```
INSERT INTO knigi VALUES ('B3498-43', 'Горе от ума',
'А. С. Грибоедов', 1996, 'Детская литература',
354, 20.50, 2016:01:02, 'Пьеса в стихах')
```

Добавляемые значения должны соответствовать тому порядку, в котором поля хранятся в таблице. Если вы хотите добавлять информацию в другом порядке, то вы должны указать этот порядок в операторе INSERT, например:

```
INSERT INTO knigi (bibl_kod, name_knigi, avtor, god_izd,
izdatelstvo, kol_stranic, cena, data_vidachi, annotacia) VALUES
('B3498-43', 'Горе от ума', 'А. С. Грибоедов', 1996, 'Детская
литература', 354, 20.50, 2016:01:02, 'Пьеса в стихах')
```

С помощью INSERT можно добавлять данные в отдельные поля, но только в том случае, если при создании таблицы в остальных полях предусмотрена возможность оставлять пустые (нулевые) значения. Иначе, этот запрос выдаст ошибку.

Другая распространенная ошибка связана с попыткой вставить в таблицу новые данные с таким же первичным ключом, который уже имеется в ней.

Извлечение данных из таблиц MySQL. Для извлечения записей из базы данных в SQL используется оператор **SELECT**. Он имеет очень много возможностей, но в нашем простейшем случае, когда используется только одна таблица, мы ограничимся только возможностью выводить всю информацию из таблицы или фильтровать информацию в зависимости от какого-либо условия.

Получить все данные из таблицы:

```
SELECT * FROM knigi
```

Получить данные о всех названиях и авторах книг из таблицы knigi:

```
SELECT name_knigi, avtor FROM knigi
```

Получить все сведения о книгах автора А. С. Грибоедов

```
SELECT * FROM knigi WHERE avtor = 'А. С. Грибоедов'
```

Получить все сведения о книгах автора Грибоедов, если мы не помним его инициалов:

```
SELECT * FROM knigi WHERE avtor LIKE '%Грибоедов%'
```

Получить все сведения о книгах, изданных в период с 1950 по 1970 гг.:

```
SELECT * FROM knigi WHERE god_izd BETWEEN 1950 AND 1970
```

Получить все сведения о книгах Грибоедова, изданных в период с 1950 по 1970 годы:

```
SELECT * FROM knigi
  WHERE (god_izd BETWEEN 1950 AND 1970)
  AND (avtor LIKE '%Грибоедов%')
```

Изменение данных в таблицах. Синтаксис оператора **UPDATE**, который используется для обновления записей, выглядит так:

```
UPDATE Имя_таблицы SET Поле1 = Значение1, ... ,
ПолеN = ЗначениеN [WHERE Условие]
```

Если не задано условие WHERE, будет модифицирована вся таблица, а это может повлечь за собой непредсказуемые последствия, поскольку для всех записей будут установлены одинаковые значения полей, поэтому, практически всегда нужно указывать условие WHERE.

Предположим, нам необходимо обновить записи о книгах, изданных до 2000 года, снизив их цену на 10 %.

Запрос выглядит так:

```
UPDATE knigi SET cena = cena*0.9 WHERE god_izd < 2000
```

Данный запрос нужно понимать так: найти запись (или записи), значение поля god_izd которой меньше 2000, и установить значение cena в ней меньше на 10 %.

Удаление данных из таблиц. Для удаления строк из таблиц в SQL используется оператор **DELETE**.

Если нам необходимо удалить все данные о книгах, изданных в издательстве «Детская литература», необходимо указать:

```
DELETE FROM knigi WHERE izdatelstvo = 'Детская литература'
```

7.4. Функции PHP для работы с MySQL

Рассмотрим основные функции PHP, применяемые для работы с базой данных MySQL.

7.4.1. Функции соединения с сервером MySQL и базой данных

Здесь предлагается использовать MySQLi (MySQL Improved) — расширение драйвера реляционных СУБД, используемого в языке программирования PHP для предоставления доступа к базам данных MySQL. MySQLi является обновленной версией драйвера PHP MySQL, и обеспечивает различные улучшения в работе с базами данных. Это расширение основано на объектно-ориентированном подходе и использует основной класс для работы с БД — mysqli.

Для создания соединения с БД нужно создать новый объект класса mysqli:

```
/*Указываем имя сервера – localhost, имя пользователя базы данных - my_user, пароль пользователя БД - my_password, имя базы данных - my_db */

$mysqli = new mysqli(
  'localhost', 'my_user',
  'my_password', 'my_db');

// Выполняем обработку ошибки подключения к базе данных
If ($mysqli->connect_error) {
    die(' Connect Error ('.$mysqli->connect_errno. ') '.$mysqli->connect_error);
}
```

С помощью *$mysqli->connect_errno* и *$mysqli->connect_error* мы получаем описание и код ошибки, возникших при соединении.

Для закрытия соединения предназначена функция mysqli_close($connection_id).

```
$mysqli->close();
```

7.4.2. Функции выполнения запросов к серверу баз данных

Все запросы к текущей базе данных отправляются функцией *mysqli->query*(). Этой функции нужно передать текст запроса.

Текст запроса модет содержать пробельные символы и символы новой строки (\n). Текст должен быть составлен по правилам синтаксиса SQL.

Пример запроса:

```
$result_set = $mysqli->query('SELECT * FROM users');
```

Приведенный запрос должен вернуть все содержимое таблицы **users**. Результат запроса присваивается переменной $*result_set*. Результат — это набор данных, который после выполнения запроса нужно обработать определенным образом.

Другой пример запроса:

```
$result_set1 = $mysqli->query(
  "INSERT INTO users (login,pass,email)
   VALUES('$login','$password','$email')");
```

позволяет вставить данные в строку таблицы users.

7.4.3. Функции обработки результатов запроса

Если запрос, выполненный с помощью функции mysqli->query() успешно выполнился, то в результате клиент получит набор записей, который может быть обработан следующими методами:

— *$result->fetch_row*() получает текущий ряд результата в виде нумерованного массива,

— *$result->fetch_assoc*() — в виде ассоциативного массива,

— *$result->fetch_array*() — тип массива задается константой.

В следующем примере был получен результат — все содержимое таблицы **users**, затем он помещен в ассоциативный массив, и по каждой строке этого массива были выведены идентификационный номер, логин и пароль пользователя. Ключи в этом случае совпадают с именами полей в таблице users.

```
$result_set = $mysqli->query('SELECT * FROM users');
  while ($row = $result_set->fetch_assoc())
  {
    echo $row[id];
    echo '<p>';
    echo $row[login];
    echo '<p>';
    echo $row[pass];
    echo '<p>';
    $var = $var + 1;
  }
```

Еще один пример использования интерфейса к базе данных с применением mysqli приведен ниже.

```
<?php
  /* Подключение к серверу MySQL */
  mysqli = new mysqli('localhost','user', 'password', 'world');
  if (mysqli_connect_errno()) {
    printf(
      "Подключение к серверу MySQL невозможно. Код ошибки: %s\n",
      mysqli_connect_error());
    exit;
  }
  /* Посылаем запрос серверу */
  if ($result = $mysqli->query('SELECT Name, Population FROM City)) {
    print("Население городов:\n");
    /* Выбираем результаты запроса: */
    while( $row = $result->fetch_assoc() ){
      echo $row['Name'] . ":" . $row['Population'] . "<br>";
    }
    /* Освобождаем память */
    $result->close();
  }
  /* Закрываем соединение */
  $mysqli->close();
?>
```

Таким образом, становится понятно, как PHP позволяет взаимодействовать с базой данных.

Необходимо отметить, что современные тенденции Web-разработки предполагают, что вывод результатов запросов к базе данных на страницу выполняется не на сервере, а на клиенте. То есть лучшие практики используют технологии, которые позволяют представлять результат обработки пользователю в браузере следующим образом:

1) на сервере размещен специальный сервис, который позволяет, с использованием одного из серверных языков программирования (например, PHP), выполнять определенный запрос к базе данных и результат этого запроса сохранять в специальном текстовом формате, например, JSON;

2) браузер загружает присланную с сервера страницу, при необходимости отобразить результат запроса обращается к соответствующему сервису, получает данные в формате JSON и размещает (иногда говорят «отрисовывает») их на странице с использованием JavaScript или более удобных его библиотек и фреймворков, таких как React.

Этот подход будет нами отработан в практической работе № 9.

Контрольные вопросы

1. Для чего используются базы данных при создании сайтов? Какие преимущества этой технологии хранения данных вы можете назвать?
2. Что такое СУБД? Какие СУБД Вам известны? Какая СУБД наиболее часто применяется для сайтов, написанных на PHP?
3. Как вы представляете себе базу данных MySQL?
4. Какие этапы включает процесс создания базы данных MySQL?
5. Какие типы могут иметь столбцы таблицы базы данных MySQL?
6. Какой язык применяется для получения информации из базы данных или изменения данных в ней?
7. Как организовать SQL запрос для вставки новых строк в таблицу БД? Приведите пример.
8. Как организовать SQL запрос для поиска строк в таблице БД? Приведите пример.
9. Как организовать SQL запрос для обновления строк в таблице БД? Приведите пример.
10. Как организовать SQL запрос для удаления строк в таблице БД? Приведите пример.
11. Опишите функцию подключения к БД MySQL из программы PHP.
12. Опишите функцию выполнения запросов к БД MySQL из программы PHP.
13. Опишите функции обработки результатов запроса к БД MySQL из программы PHP.
14. В чем суть использования технологий клиенской обработки запросов к базе данных?

Контрольные задания

1. Опишите структуру таблицы БД, которая хранит сведения о статье: идентификационный номер статьи, автор статьи, название статьи, дата создания статьи, содержание статьи.
2. Опишите таблицу из задания 1 в терминах СУБД MySQL: придумайте наименования, типы и возможные ключи для каждого из полей таблицы.
3. Составьте запрос на языке SQL, позволяющий добавлять сведения о новой статье к таблице из задания 1.
4. Составьте запрос на языке SQL, позволяющий обновлять содержимое некоторой статьи в таблице из задания 1.
5. Составьте запрос на языке SQL, позволяющий удалять данные некоторой статьи в таблице из задания 1.
6. Составьте запрос, отбирающий для будущего отображения на странице некоторую статью из таблицы из задания 1.
7. Составьте программу на языке PHP, которая позволяет:

а) подключиться к базе данных, расположенной на сервере с именем $a под логином $b, паролем $c, к базе данных под именем $d;

б) выполнить запрос по получению некоторой статьи из базы данных;

в) разместить эту статью на странице с выводом информации об авторе, наименовании, дате создания и содержании статьи.

Тема 8
MVC-ФРЕЙМВОРКИ И CMS-СИСТЕМЫ

После изучения данного раздела студент должен

знать

- назначение модели MVC,
- основные составляющие модели MVC и их назначение,
- различные технологии создания динамических web-приложений,
- понятие и назначение PHP-фреймворков,
- понятие и назначение CMS-систем,
- преимущества и недостатки использования различных технологий разработки web-приложений;

уметь

- использовать одну из современных CMS-систем для создания web-приложений;

владеть

- навыками сравнительного анализа современных фреймворков и CMS-систем для обоснования выбора технологии разработки web-приложений.

8.1. Понятие MVC

На сегодняшний день можно выделить два наиболее типичных способа создания динамических web-страниц.

Первый способ предполагает, что в одном файле может содержаться код как на языках программирования, так и на языках разметки. Например, в начале файла производится запрос к базе данных для получения из нее какой-либо информации. Здесь, как мы видели ранее, используются методы PHP с внедренными в них запросами на языке SQL. После этого, как правило, начинается HTML-разметка страницы. В нужных местах производятся вставки PHP-кода, которые обрабатывают результаты SQL-запроса и «отдают» его в виде HTML-разметки. Таким образом, в одном файле применяются: SQL, HTML и PHP. При

этом оформление страниц может включать стили CSS и, при необходимости, JavaScript.

Понятно, что такой стиль программирования затрудняет понимание логики, усложняет внесение изменений и требует постоянного участия создателя сайта.

Второй способ связан с применением шаблона проектирования ***MVC*** (*Model-View-Controller*, Модель-Представление-Контроллер).

Основная идея данного подхода заключается в необходимости распределения однородных элементов по разным файлам. Если говорить очень упрощенно: один файл — один язык. Помимо идеи разнесения разных языков в разные файлы, ключевой концепцией является также разделение файлов на группы в соответствии с теми функциями, которые они выполняют в приложении.

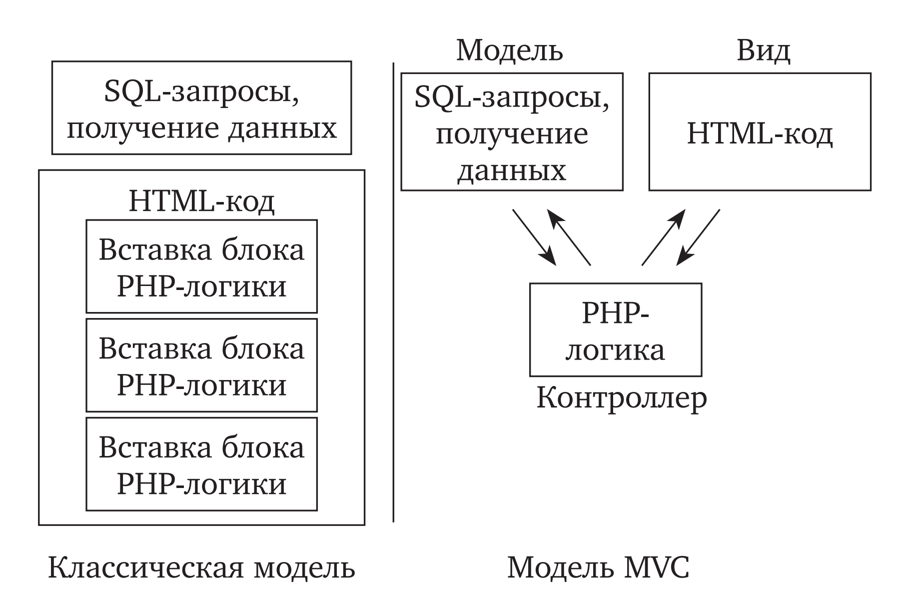


Рис. 8.1. **Отличия схемы MVC и традиционной схемы программирования сайтов**

Шаблон проектирования MVC предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Модель представляет собой совокупность процедур и алгоритмов обработки данных. Сама по себе Модель не содержит данных, но как правило получает их из БД и обрабатывает по заранее описанным алгоритмам. Если говорить о web-

разработке, то модель будет содержать набор классов и функций, например, на языке PHP, которые реализуют работу с базой данных и интерпретируют результаты SQL-запросов.

Естественно, чтение информации из БД может быть выполнено несколькими функциями. В качестве примера можно рассмотреть модель, реализующую работу с базой данных статей для сайта. Отдельные функции помогут получить конкретную статью из БД, список последних статей, список популярных статей, список статей, относящихся к определенной теме и т. д.

Второй элемент — это ***Представление, View***. Оно позволяет отобразить информацию. Если это сайт, то информация отображается в браузере. Представление при разработке сайтов содержит HTML-код, в который подставляются переменные, которые берутся не из модели, а из контроллера. Представление отслеживает изменения в модели и создает или меняет интерфейс PHP-приложения.

Третий элемент — это ***Контроллер.*** Его главная функция — это обеспечение связи между представлением и моделью. Он также может содержать PHP-код. При обращении пользователя по нужному URL выбирается соответствующий контролер, который обращается к модели, после чего информация выводится посредством нужного представления.

8.2. Использование PHP-фреймворков

В первой части данного учебного пособия уже говорилось о фреймворках (в частности о CSS-фреймворках). Фреймворк для любой из серверных технологий web-программирования — это фундамент, который определяет архитектуру будущего приложения и содержит в себе отлаженный код для решения часто используемых задач web-разработчика, таких как работа с формами, базой данных, шаблонами и т. д. На базе фреймворка можно написать сложную прикладную информационную систему или даже конструктор других систем.

PHP-фреймворк — это набор технологий, которые включают не только библиотеки распространенных модулей, но и шаблоны проектирования. Можно при написании каждого нового приложения изобретать велосипед с распределением его основных модулей, структурой папок, классами обработки основных компонентов и т. п., а можно воспользоваться готовым универ-

сальным решением. Большинство таких фреймворков используют модель MVC, что позволяет создавать более читабельный код, который легче изменять. Большинство PHP-фреймворков содержат встроенные средства выполнения основных операций с базами данных.

Ниже представлен обзор некоторых наиболее широко используемых в наше время PHP-фреймворков.

Laravel. Один из самых популярных PHP-фреймворков, позволяющий максимально упростить решение основных задач, таких как аутентификация, маршрутизация, сессии и кэширование. Laravel создавался как попытка объединить все лучшее, что есть в других PHP-фреймворках, а также в технологиях ASP. NET, MVC и других. Одно из самых важных его достоинств — наличие интегрированной системы модульного тестирования.

Zend. Объектно-ориентированный фреймворк, использующий все последние достижения PHP. Разработан так, что каждый компонент можно использовать отдельно. Стандартный набор библиотек делает его очень мощным и легко расширяемым средством разработки. Кроме того, он предлагает надежную и высокопроизводительную реализацию MVC и удобную в использовании абстракцию базы данных, а также множество других возможностей, которые делают его одним из самых функциональных фреймворков.

CakePHP — написанный на PHP программный каркас для создания веб-приложений, поддерживаемый быстрорастущим сообществом. Как и большинство других фреймворков, реализует шаблон проектирования MVC.

Code Igniter. Наиболее простой в освоении и использовании фреймворк. Легко расширяется, безопасен и использует простые и понятные подходы. Одно из основных его преимуществ — скорость работы, он быстрее справляется с задачей взаимодействия с БД, чем другие.

Symfony. Одно из основных его достоинств — поддержка множества СУБД (MySQL, PostgreSQL, SQLite).

Yii. Высокопроизводительный PHP-фреймворк, чьи возможности позволяют в сжатые сроки реализовывать крупномасштабные проекты типа форумов, порталов, CMS (см. ниже) и других сложных систем. В него встроено множество проверенных и готовых к использованию решений: конструктор запросов, многоуровневая поддержка кэширования и многие другие.

8.3. Понятие и возможности CMS

Выше были описаны два пути создания динамических web-сайтов. Во-первых, это написание программ на одном из языков, например, PHP, «с нуля», во-вторых — использование готовых решений для реализации основных функциональных модулей сайтов с применением фреймворков.

Существует еще один путь — воспользоваться программными средами, которые называются ***системами управления контентом*** **(*Content Management System, CMS*).** Преимуществом этого пути является уменьшение затрат времени — создавать сайты с использованием CMS могут даже не очень опытные разработчики. К недостаткам применения CMS можно отнести снижение гибкости, предоставление недостаточного или слишком большого набора возможностей.

Под ***контентом*** (от англ. *content*, содержание или содержимое) понимают информационное наполнение сайта — то есть все типы материалов, которые находятся на сервере: web-страницы, документы, программы, аудиофайлы, фильмы и так далее. Таким образом, управление контентом — это процесс управления подобными материалами. Этот процесс включает следующие действия: размещение материалов в базе данных, удаление материалов из базы данных, организацию (реорганизацию) материалов, возможность отслеживать их состояние.

Функции CMS можно разделить на несколько основных категорий.

1. *Создание* — предоставление авторам удобных и привычных средств создания контента.

2. *Управление* — хранение контента в едином репозитории. Это позволяет следить за версиями документов, контролировать, кто и когда их изменял, убеждаться, что каждый пользователь может изменить только тот раздел, за который он отвечает. Кроме того, обеспечивается интеграция с существующими информационными источниками и ИТ-системами.

3. *Публикация* — автоматическое размещение контента на сайте пользователя. Соответствующие инструменты автоматически адаптируют внешний вид страницы к дизайну всего сайта.

4. *Представление* — дополнительные функции, позволяющие улучшить форму представления данных; например, можно строить навигацию по структуре репозитория.

Использование CMS предоставляет следующие преимущества:

1. *Оперативное обновление информации* — информацию публикует сотрудник, владеющий информацией, без дополнительных посредников в виде технических специалистов. Пользователь сам может определять визуальное представление своих материалов, используя для этого стандартные средства, не требующие знания языка HTML и других достаточно сложных для неспециалиста процедур.

2. *Снижение стоимости поддержки* — обновление информации производится сотрудниками самостоятельно, нет необходимости оплачивать труд собственного или внешнего web-мастера. Снижение стоимости происходит за счет снижения потерь времени на поиски документов, пресечения дублирования и ошибок, увеличения скорости связи с партнерами и клиентами.

3. *Предоставление дополнительных сервисов пользователю* — часть сервисов (поиск, форумы, голосования и т. д.) требует интерактивного взаимодействия с пользователем. Они уже реализованы в рамках CMS.

4. *Уменьшение сроков и стоимости разработки* — наиболее востребованная функциональность уже реализована в CMS и может быть сразу использована.

5. *Повышение качества разработки* — при разработке полностью или частично используются готовые модули, которые уже прошли неоднократное тестирование.

6. *Снижение стоимости дальнейших модификаций* — CMS позволяют разделить данные и их представление. Это позволяет гораздо проще изменить внешний вид сайта, чем в случае со статическим сайтом.

Рассмотрим наиболее распространеные современные CMS, основанные на применении PHP.

Drupal — это не только система управления контентом в классическом понимании этого термина. Это больше ядро, на основе которого можно собрать практически неограниченную функциональность сайта.

На CMS Drupal построены тысячи сайтов. Основные направления разработок на Drupal — это блоги, социальные сети, персональные сайты, корпоративные сайты, порталы сообществ, форумы, магазины, сайты-справочники.

Drupal считается одной из самых надежных систем на сегодняшний день. Ее важное преимущество — бесплатность использования. Есть русскоязычное сообщество Drupal, где можно получить нужную техническую помощь.

К недостаткам использования Drupal следует отнести сложность в освоении и использовании. Человек без начальных знаний web-программирования вряд ли сможет использовать мощную функциональность системы.

Joomla позволяет создавать сайты-визитки, интернет-магазины, порталы, сообщества, системы блогов, доски объявлений, корпоративные мультиязычные сайты и многие-многие другие сайты.

Joomla полностью бесплатна, обладает множеством модулей расширения функционала (они имеют разные названия: «модули», «компоненты», «мамботы», и направлены на решение различных задач). С помощью этих модулей можно добавить интернет-магазин на сайт компании буквально за несколько минут. При этом заказчик получает надежный магазин, который будет стабильно работать. Создано огромное число готовых шаблонов, которые подойдут практически для любого сайта, можно также разработать и применить ко всем страницам собственный шаблон, хотя это требует определенного уровня знаний в области верстки и программирования.

WordPress. Бесплатная CMS, которая еще недавно рассматривалась только как инструмент для создания блогов. Однако, функциональность WordPress, как и всех остальных CMS, расширяется установкой дополнительных плагинов. Отличается очень простым алгоритмом установки и настройки, в настоящее время активно развивается.

В практической части курса мы рассмотрим основные возможности CMS WordPress.

Контрольные вопросы

1. Для чего используется модель MVC?
2. Опишите назначение основных элементов модели MVC?
3. Что такое фреймворк?
4. Для чего используются PHP-фреймворки?
5. Какие PHP-фреймворки Вам известны?
6. Что такое CMS?
7. Какие типы CMS-систем Вам известны?

8. Какие системы управления контентом сайтов Вам известны?
9. В чем отличие технологии создания сайта вручную (путем программирования всех страниц), с применением фреймворков и с применением CMS?
10. Какие достоинства и недостатки вы видите в применении перечисленных в вопросе 9 технологий реализации web-сайтов?

ПРАКТИКУМ

Практическая работа № 1
ОСНОВЫ ЯЗЫКА HTML

Цель: Изучить основные возможности языка HTML для создания и связывания web-документов.

Инструментарий: 1) любой редактор программного кода, например, notepad++, который можно бесплатно скачать с сайта разработчика по адресу https://notepad-plus-plus.org/downloads/; 2) любой современный браузер.

Задания

1. Выполнить все шаги практической работы, в результате которых должен получиться трехстраничный сайт-визитка туристической компании.

2. По аналогии реализовать свой сайт, используя только возможности языка HTML, на любую интересующую вас тему. Меню вашего сайта должно быть вертикальным. Необходимо задействовать все известные вам типы тегов: заголовки, абзацы, списки, изображения, таблицы, формы.

Выполнение задания 1

Шаг 1. Создание главной страницы сайта.

На этом шаге мы показываем реализацию в среде Notepad++. Следующие шаги описываются только кодами страниц, которые вы также можете вводить и отлаживать в Notepad++.

Итак, в среде Notepad++ создадим новый документ и сохраним его в отдельной папке под именем index.html.

Введем следующий текст. Старайтесь соблюдать структуру документа — отступы с использованием пробелов или табуляции, которые позволяют описать иерархию тегов.

```
<!-- Начало HTML документа -->
<!DOCTYPE html>
<html>
<!-- Начало заголовка документа -->
<head>
```

```
  <!-- Служебная информация о кодировке документа -->
  <meta http-equiv = "Content-Type"
    content = "text/html; charset = utf-8">
  <!-- Информация, которая будет отображаться на вкладке браузера
-->
  <title>
    Сайт туристической компании
  </title>
<!-- Конец заголовка документа -->
</head>
<!-- Начало тела документа -->
<body>
<!-- Начало блока, содержащего всю страницу -->
  <div id = "content">
    <!-- Начало блока заголовка страницы -->
    <header>
      <!-- Заголовок страницы -->
      <h1>
        ПУТЕШЕСТВУЙ С НАМИ!
      </h1>
    <!-- Конец блока заголовка страницы -->
    </header>
    <!-- Начало блока меню страницы -->
    <nav>
      <!-- Гиперссылка на главную страницу сайта -->
      <a href = "index.html">
        Главная страница
      </a>
      <!-- Гиперссылка на страницу с информацией о турах -->
      <a href = "turs.html">
        Наши туры
      </a>
      <!-- Гиперссылка на страницу с контактной информацией -->
      <a href = "contacts.html">
        О нас
      </a>
    <!-- Конец блока меню -->
    </nav>
    <!-- Начало блока с основным содержимым страницы -->
    <div id = "main">
      <!-- Начало нового абзаца -->
      <p>
        Концепция отдыха с нашей компанией - это отдых в новом
формате, где учтены интересы всех поколений.
      </p>
      <p>
        Наши основные направления:
        <!-- Начало блока с перечнем пунктов -->
        <ul>
          <!-- Блоки  перечня пунктов -->
```

```
        <li>Крым</li>
        <li>Кавказ </li>
        <li>Алтай</li>
      </ul>
    </p>
    <p>
      Акции и скидки
    </p>
  <!-- Конец блока с основным содержимым -->
  </div>
  <!-- Начало блока "подвала" страницы -->
  <footer>
    Это сайт, предназначенный для обучения
  <!-- Конец блока "подвала" страницы -->
  </footer>
<!-- Конец блока с полным содержимым страницы -->
</div>

<!-- Конец тела документа -->
</body>
<!-- Конец документа -->
</html>
```

Если открыть этот файл в любом из браузеров, должен получиться примерно такой результат:

ПУТЕШЕСТВУЙ С НАМИ!

Главная страница Наши туры О нас

Концепция отдыха с нашей компанией - это отдых в новом формате, где учтены интересы всех поколений

Наши основные направления:

- Крым
- Кавказ
- Алтай

Акции и скидки

Это сайт, предназначенный для обучения

Шаг 2. Описание структуры страницы.

Попытаемся отформатировать страницу так, чтобы ее блоки занимали определенные области на экране за счет применения стилей к каждому из блоков.

В область <head>…</head> необходимо добавить следующий код:

```
<!-- Начало описания стилей --!>
<style>
    /*-- Описание стиля всей страницы --*/
    div#content {
```

```
    /*-- Ширина - 80% от всего экрана --*/
    width: 80%;
    /*-- Рамка отсутствует --*/
    border: none;
    /*-- Выравнивание по центру занимаемой области --*/
    margin: auto;
  }

  /*-- Описание стиля заголовка и подвала --!>
  header, footer {
    /*-- Отступ текста от края блока - 1em - текущий размер
шрифта, для увеличения или уменьшения можно брать любые пропорции
от текущего:2em,0.5em --*/
    padding: 1em;
    /*-- Цвет шрифта-белый --*/
    color: white;
    /*-- Цвет фона --*/
    background-color: #007196;
    /*-- Отменяет обтекание с левого края элемента,
все другие элементы на этой стороне будут располагаться
под текущим элементом --*/
    clear: left;
    /*-- Выравнивание текста по центру --*/
    text-align: center;
  }

  /*-- Описание стиля блока меню --*/
  nav {
    /*-- Высота блока в пикселях --*/
    height: 40px;
    /*-- Ширина и цвет линии обрамления --*/
    border: solid #333;
    padding: 1em;
    text-align: center;
  }

  /*-- Описание стиля ссылки в меню --!>
  nav a {
    /*-- Для ссылок используется оформление по умолчанию --*/
    text-decoration: none;
    padding: 1em;
  }

  /*-- Описание стиля основного блока страницы --*/
  #main {
    padding: 1em;
  }
</style>
```

Результат этих изменений в браузере:

ПУТЕШЕСТВУЙ С НАМИ!

Главная страница Наши туры О нас

Концепция отдыха с нашей компанией - это отдых в новом формате, где учтены интересы всех поколений

Наши основные направления:

- Крым
- Кавказ
- Алтай

Акции и скидки

Это сайт, предназначенный для обучения

Шаг 3. Добавление оформления к элементам таблицы.

В папке, содержащей файл index.html, создадим папку img для хранения картинок, которые будут использоваться на сайте.

Скопируем в нее картинку по тематике сайта, и пусть, например, она имеет имя logo.jpg.

Поместим картинку logo.jpg слева от заголовка сайта. Для этого в блоке <header>…</header> нужно создать два блока — один для вывода логотипа, другой — для названия. Соответствующие изменения нужно внести в описание стилей.

Изменения в структуре:

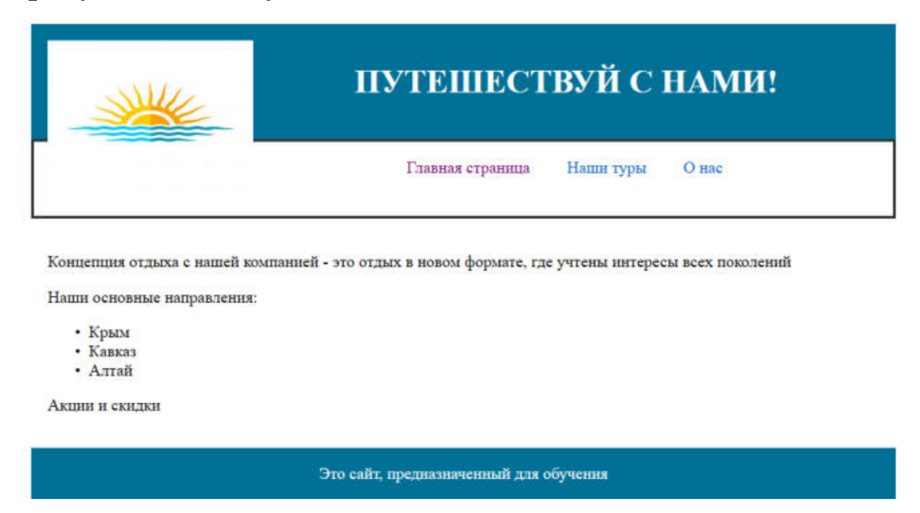
```
<header>
  <!-- Начало блока для логотипа -->
  <div id = "logo">
    <!--Вывод логотипа-->
    <img src = "img/logo.jpg">
  <!-- Конец блока для логотипа -->
  </div>
  <!-- Начало блока для названия -->
  <div id = "name">
    <h1>ПУТЕШЕСТВУЙ С НАМИ!</h1>
  <!-- Конец блока для названия -->
  </div>
</header>
```

Добавление к стилям:

```
/*-- для тега img, который размещен в блоке с идентификатором
logo: ширина- 200 пикселей, высота- 150 пикселей, прижать к левой
части родительского блока и заставить следующий блок обтекать его
справа --*/
```

```
div #logo img {
  width: 200px;
  height: 150px;
  float: left;
}
```

В результате получим:



Шаг 4. Использование полученного шаблона для создания других страниц сайта.

Создайте две копии полученной страницы в той же папке и дайте им имена turs.html и contacts.html. Именно так, если вы помните, называются файлы, на которые уже имеются ссылки в нашем меню. Теперь для поддержания общей структуры и оформления сайта на любой странице мы будем изменять только содержимое страницы, которое размещается внутри блока с идентификатором ***main***.

Шаг 5. Создание содержимого второй страницы.

На второй странице с именем turs.html создадим таблицу, содержащую список туров. Это можно было бы сделать с использованием блоков div, но иногда полезно применять таблицы, и мы потренируемся делать это. Итак, пусть необходимо создать таблицу, содержащую три строки с названиями туров, каждая из которых содержит также фото и некоторую дополнительную информацию

В блок main поместим следующий код:

```
<div id = "main">
  <!-- Начало таблицы с идентификатором tours -->
  <table id = "tours">
```

```
    <!-- Первая строка таблицы -->
    <tr>
      <!-- Первый столбец первой строки будет далее делится
на два -->
      <td rowspan = 2>Туры в Крым</td>
      <!-- Второй столбец первой строки -->
      <td>Отель "Волна"</td>
    </tr>
    <!-- Вторая строка -->
    <tr>
      <!-- Второй столбец второй строки, первый столбец
"растянулся" из первой строки -->
      <td>Отель "Море"</td>
    </tr>
    <!-- Третья строка -->
    <tr>
      <td rowspan = 2>Туры на Кавказ</td>
      <td>Курортный отдых</td>
    <tr>
      <td>Горный туризм</td>
    </tr>
    <tr>
      <td rowspan = 2>Туры на Алтай</td>
      <td>Зеленый туризм</td>
    <tr>
      <td>Конные туры</td>
    </tr>
  </table>
</div>
```

Добавим стили для таблицы:

```
/*-- Для всей таблицы: --*/
table#tours {
  /*-- Ширина 80% от ширины родительского блока --*/
  width:80%;
  /*-- Выравнивание по центру --*/
  margin:auto;
}

/*-- Для отдельной ячейки таблицы: --*/
table#tours td{
  /*-- Текст выровнен по левому краю --*/
  text-align: left;
  /*-- Сплошная красная рамка толщиной 2 пикселя --*/
  border: 2px solid red;
}
```

Получим:



Шаг 6. Создание форм.

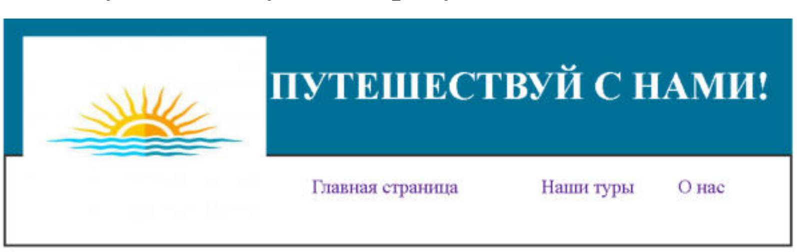
На примере страницы contacts.html продемонстрируем использование форм. Пусть на третьей странице сайта необходимо создать форму для обращения пользователя к администратору сайта.

Для более аккуратного вывода форму можно также разместить в виде таблицы. Для этого в блок main третьей страницы добавим следующий код:

```
<p>
  Наш адрес: г. Белогорск, ул. Ракетная, 18.
</p>
<p>
  Для связи с нами заполните форму:
</p>
<!-- Начало формы, программа, которая будет ее обрабатывать пока
неизвестна -->
<form method = "get" action = "#">
<!-- Начало таблицы -->
<table id = "form">
  <tr>
    <td>Введите свое имя</td>
    <!-- Простое текстовое поле для ввода имени -->
    <td><input name = "name"></td>
  </tr>
  <tr>
    <td>Введите ваш номер телефона </td>
    <!-- Простое текстовое поле для ввода телефона -->
    <td><input name = "tel"></td>
  </tr>
```

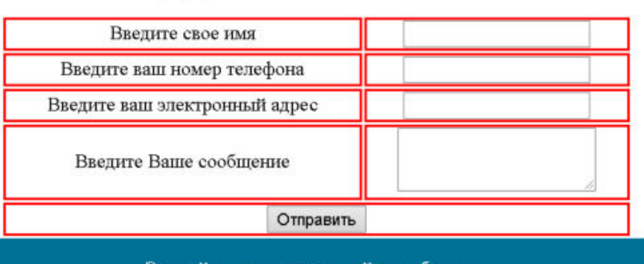
```
  <tr>
    <td>Введите ваш электронный адрес</td>
    <!-- Простое текстовое поле для ввода адреса электронной почты
-->
    <td><input name = "mail"></td>
  </tr>
  <tr>
    <td>Введите Ваше сообщение</td>
    <td>
      <!-- Поле для ввода многострочного сообщения -->
      <textarea name = "text" rows = "3">
          </textarea>
    </td>
  </tr>
  <tr>
    <td colspan = 2>
      <!-- Кнопка для отправки формы на обработку -->
      <input name = "otp" type = "submit" value = "Отправить">
    </td>
  </tr>
</table>
```

Если указать стиль для этой таблицы, такой же как для предыдущей, получим следующий результат:



Наш адрес: г. Белогорск, ул. Ракетная, 18

Для связи с нами заполните форму:



Это сайт, предназначенный для обучения

Практическая работа № 2
ОСНОВЫ ИСПОЛЬЗОВАНИЯ CSS

Цель: Изучить основные возможности технологии CSS для создания структуры web-документов и единого стиля оформления web-сайтов.

Инструментарий: 1) Любой редактор программного кода, например, notepad++; 2) Любой современный браузер.

Задания

1. Создать внешнюю таблицу стилей для сайта, разработанного в ходе выполнения задания 1 из практической работы № 1, которая позволит продемонстрировать возможности CSS по разметке и единому оформлению web-сайта.

2. Усовершенствовать собственный сайт, который был реализован в задании 2 практической работы № 1. В результате должен получиться сайт с вертикальным интерактивным меню с кнопками, оформленный в едином стиле, содержащий список новостей с иллюстрациями на главной странице, дополнительную информацию на второй странице и контакты с возможностью обратной связи на третьей странице. Страницы сайта должны содержать только HTML-код, все стили должны быть вынесены в файл style.css, который помещен в папку CSS в папке проекта.

Выполнение задания 1

Все шаги выполняются для сайта туристической компании, который был создан в ходе выполнения задания № 1 из Практической работы № 1.

Шаг 1. Создание и подключение внешней таблицы стилей.

В папке, содержащей созданный в ходе выполнения практической работы № 1 проект, создайте папку CSS.

Внутри этой папки создайте текстовый файл style, который сохраните с расширением style.css. Переместите в этот файл содержимое блока <style>…</style> из файла index.html (без тегов <style>…</style>).

Удалите блок <style>...</style> из всех HTML-файлов проекта.

Вместо этого добавьте в каждый из файлов в область <head>...</head> строку подключения таблицы стилей:

```
<link href = "css/style.css" rel = "stylesheet" type = "text/css">
```

Убедитесь в том, что сайт по-прежнему поддерживает все созданные ранее стили.

Теперь для изменения стилей всего сайта будем править только таблицу стилей.

Шаг 2. Работа с общей таблицей стилей.

Изменим стиль шрифта всего сайта. Для этого укажем (дополнительно к уже имеющимся стилям):

```
/* Для всего содержимого страницы */
div#content {
  /* Начертание шрифта */
  font-family:Verdana;
  /* Стиль шрифта */
  font-style:normal;
  /* Размер шрифта */
  font-size:normal;
  /* Жирность шрифта */
  font-weight:bold;
  /* Выравнивание текста */
  text-align:center;
}
```

Все возможные значения этих и других атрибутов шрифта можно увидеть, например по этой ссылке: http://htmlbook.ru/content/svoystva-teksta.

Коды цветов можно получить пройдя по этой, например, ссылке: https://colorscheme.ru/html-colors.html.

Однако если вы посмотрите на полученный результат, то становится очевидно, что жирный (bold) шрифт в основной области страницы не нужен. Возможно также, лучше изменить выравнивание текста в этой области на выравнивание по всей ширине. Для этого мы можем *переопределить* это свойство в стиле блока с идентификатором main:

```
#main {
  /* Новый цвет текста содержимого */
  color:#008080;
  /* Отмена жирного шрифта */
  font-weight:normal;
```

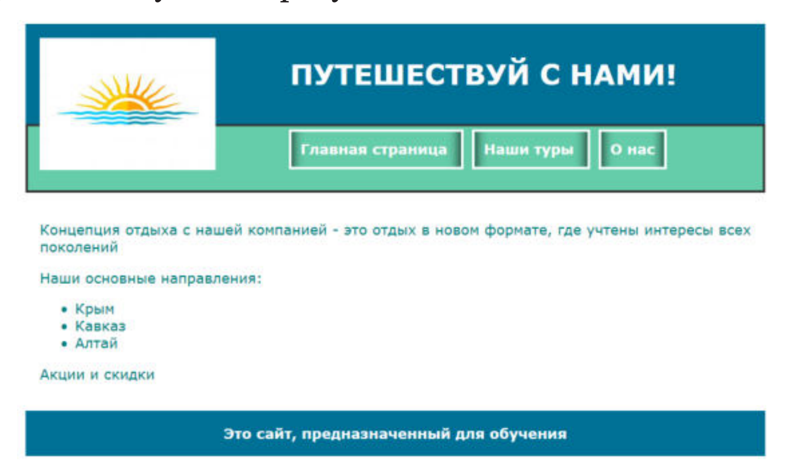
```
  /* Выравнивание по всей ширине блока */
  text-align:justify;
}
```

Далее, хотелось бы, чтобы ссылки в меню были больше похожи на кнопки. Для этого их можно оформить, например, так:

```
/* для всей области меню */
nav {
  height: 40px;
  border: solid #333;
  padding: 1em;
  text-align: center;
  list-style-type:none;
  /* цвет фона меню */
  background-color:#66CDAA;
}

/* для отдельных ссылок в области меню */
nav a {
  /* отступ между пунктами меню */
  margin-right:10px;
  /* отменить подчеркивание ссылки */
  text-decoration:none;
  padding: 10px;
  color:white;
  /* рамка для каждого пункта меню */
border:solid white 2px; }
```

Получим следующий результат:



Шаг 4. Применение стилей для классов блоков.

Теперь попробуем разбить содержимое главной области еще на три блока, в каждом из которых будет выводиться информация об одном из направлений путешествий. Поскольку все об-

ласти должны быть оформлены в одном и том же стиле, удобно применять классы.

Сохраним в папке проекта img три изображения, соответствующие трем направлениям, о которых хотим написать: crimea.jpg, kavkaz.jpg, altay.jpg.

Добавим в файл index.html такой фрагмент кода:

```
<!-- Создадим блок-обертку для всех блоков по направлениям -->
<div id = "tours">
  <p>Акции и скидки</p>
  <!-- Создаем новый блок класса tour -->
  <div class = "tour">
    <p>Крым</p>
    <!-- Вставляем соответствущую картинку -->
    <img src = "img/crimea.jpg" >
    <p>
      <!--Добавляем ссылку на страницу (будущую) с более подробной
информацией -->
      <a class = "actia" href = "#" >
      Подробнее об акции
      </a>
    </p>
  </div>
  <!-- Создаем еще один блок такого же класса -->
  <div class = "tour">
    <p>Кавказ</p>
    <img src = "img/kavkaz.jpg">
    <p>
      <a class = "actia" href = "#">
      Подробнее об акции
      </a>
    </p>
  </div>
  <!-- Создаем еще один блок такого же класса -->
  <div class = "tour">
    <p>Алтай</p>
    <img src = "img/altay.jpg" >
    <p>
      <a class = "actia" href = "#">
      Подробнее об акции
      </a>
    </p>
  </div>
  <!-- Закрываем блок-обертку -->
</div>
```

Добавим к таблице стилей стили, описывающие новые блоки и их отдельные элементы:

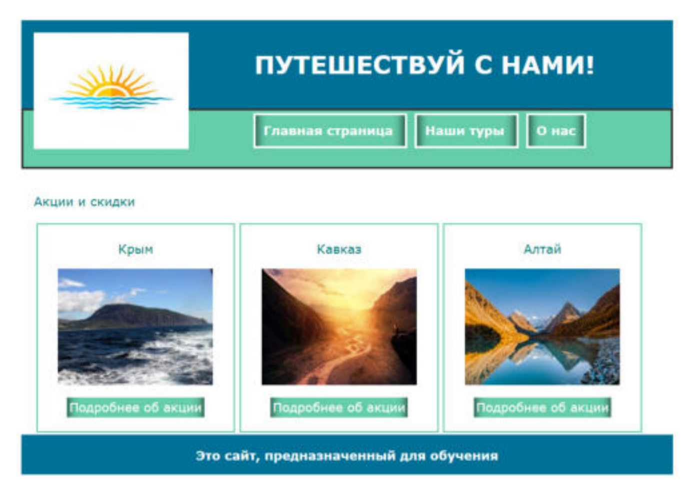
```
/* Для блока-обертки: */
#tours {
```

```
  /* Выравнивание текста по левому краю */
  text-align:left;
}
/* Для блоков каждого из направлений: */
.tour {
  /* Занимает 30% от ширины блока-обертки */
  width: 30%;
  /* В цветной рамке */
  border: #66CDAA solid 2px;
  /* Расположение текста по центру */
  text-align: center;
  /* Внутренний отступ со всех сторон */
  padding: 5px;
  /* Внешний отступ со всех сторон */
  margin: 3px;
  /* Прижимается к левой стороне, следующий блок обтекает его
справа */
  float: left;
}

/* Для картинок внутри блоков */
.tour img {
  Width-max;
  height: 150px;
}

/* Для ссылок внутри блоков (чтобы сделать их более похожими
на кнопки, в одном стиле с кнопками меню */
div.tour a {
  background-color: #66CDAA;
  text-decoration: none;
  padding: 3px;
  color: white;
  /* Кнопки с эффектом выпуклости */
  box-shadow: 20px 0 20px -20px #000 inset, -20px 0 20px -20px
#000 inset;
}
```

В результате получим:



Внешний вид страницы уже довольно привлекательный, однако при попытке просмотреть эту страницу на меньшем экране вы заметите множество проблем. Эти проблемы мы попробуем решить в следующей работе.

Практическая работа № 3
ВЕРСТКА САЙТА С ПРИМЕНЕНИЕМ ФРЕЙМВОРКА BOOTSTRAP

Цель: Изучить основные возможности технологии bootstrap для создания адаптивных web-сайтов.

Инструментарий: 1) любой редактор программного кода, например, notepad++; 2) любой современный браузер

Задания

1. Переверстать сайт, выполненный вручную в первых двух практических работах, с применением фреймворка Bootstrap.
2. Переверстать собственный web-сайт с применением того же подхода.

Выполнение задания 1

Мы создадим еще одну, совершенно новую версию сайта.

Шаг 1. Подключение и настройка Bootstrap.

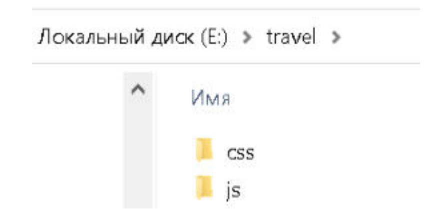
Для подключения Bootstrap можно использовать два пути.

1. Скачать фреймворк с официального сайта и хранить его вместе с проектом локально. В этом случае можно вести разработку в режиме off-line.
2. Подключить фреймворк on-line.

Мы воспользуемся первым способом.

По ссылке https://getbootstrap.com/docs/4.4/getting-started/download/ скачаем архив фреймвока (на момент написания пособия была доступна версия 4.4.1).

Создайте новую папку под именем, например, travel, и распакуйте в нее скачанный архив. При этом вы должны в папке travel получить следующую структуру папок:



В той же папке создайте файл index.html, в который скопируйте базовый документ **Starter template**, размещенный на странице **get started** сайта Bootstrap.

Открыть этот файл в используемом редакторе кода и подключить bootstrap локально:

```
<!doctype html>
<html lang = "en">
  <head>
    <!-- Required meta tags -->
    <meta charset = "utf-8">
    <meta name = "viewport" content = "width = device-width,
initial-scale = 1, shrink-to-fit = no">

    <!-- Bootstrap CSS -->
    <! Подключаем bootstrap локально -- >
    <link rel = "stylesheet" href = "css/bootstrap.min.css">

    <title>Hello, world!</title>
  </head>
  <body>
    <h1>Hello, world!</h1>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <! Подключаем библиотеки JavaScript локально -- >
    <script src = "js/bootstrap.min.js" </script>
  </body>
</html>
```

После просмотра полученного документа в браузере должно получиться следующее:



Нам придется также переопределять некоторые стандартные стили Bootstrap, поэтому создадим еще один собственный файл стилей style.css в папке CSS, которая создалась при распаковке bootstrap, и подключим его в файле index.html:

```
<!-- Bootstrap CSS -->
<link rel = "stylesheet" href = " bootstrap-4.4.1-dist/css/
bootstrap.min.css" >
...
<!-- Здесь добавляем ссылку на собственные стили -->
```

```
<!--Собственные стили--!>
<link rel = "stylesheet" href = "css/style.css" >
...
```

Если теперь в этом файле переопределить основное начертание шрифта, например так:

```
body {
  font-family:Verdana;
}
```

вы должны увидеть соответствующее изменение надписи Hello world!

Шаг 2. Использование сетки bootstrap для описания заголовка сайта.

Мы будем придерживаться стиля сайта, который был создан в первых двух работах, поэтому первой областью должна стать область заголовка. Используем для него сетку Bootstrap, которая позволяет использовать уже готовые (описанные в самой библиотеке Bootstrap) стили для строк и столбцов.

Предварительно подготовьте в папке проекта папку img с картинками из предыдущего варианта сайта

Вместо блока, который выводит «Hello Word!», поместим следующий код:

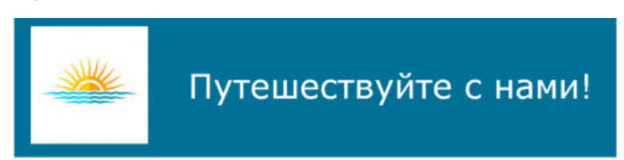
```
<!-- Используем стиль body-top по умолчанию для всего документа
-->
<body class = "body-top">
  <!-- Создаем блок для области заголовка -->
  <header class = "container">
    <!-- Создаем новую горизонтальную область (строку) -->
    <div class = "row">
      <!-- Внутри горизонтальной области определяем первую колонку
шириной 3/12 -->
      <div class = "col-3">
        <!-- Внутри выводим логотип -->
        <img id = "logo" src = "img/logo.jpg">
        <!-- Завершаем первый столбец -->
      </div>
      <!-- Внутри горизонтальной области определяем вторую
колонку, шириной 9/12 -->
        <div class = "col-9">
        <!-- Внутри выводим название -->
      <h1>Путешествуйте с нами!</h1>
      <!-- Завершаем второй столбец -->
    </div>
    <!-- Завершаем строку -->
```

```
    </div>
    <!-- Завершаем описание области заголовка -->
  </header>
  ...
```

В *своем* файле style.css переопределим стили:

```
/* Оформление содержимого всей страницы по умолчанию скопировано
с предыдущей версии сайта */
.body-top {
  width: 80%;
  border: none;
  margin: auto;
  font-family:Verdana;
  font-style: normal;
  font-size: normal;
  font-weight: bold;
  text-align: center;
}
/* Оформление содержимого всего блока заголовка */
header {
  padding: 1em;
  color: white;
  background-color: #007196;
  vertical-align: center;
}
/* Размер логотипа для средних и больших экранов */
#logo {
  max-width: 180px;
  max-height: 140px;
}
/* "Плавающее" в зависимости от изменения экрана название */
.col-9 {
  display: flex;
}
/* Заголовок в плавающем окне по центру */
h1{
  margin: auto;
}
/* МЕДИА-ЗАПРОС позволяет сделать размер логотипа и названия мень-
шими при изменении размера окна до размера мобильного устройства */
@media screen and (max-width: 768px)
{
  #logo {
    max-width: 100px;
    max-height: 70px;
  }
  h1 {
    font-size: 14px;
  }
}
```

Результат будет таким:



Обязательно попробуйте уменьшить размер экрана и убедитесь в том, что область заголовка адаптивна, то есть медиа-запрос работает.

Шаг 3. Создание меню навигации.

Это один из готовых блоков Bootstrap, код которого нужно скопировать на сайте, вставить в нужное место страницы и отредактировать стили по умолчанию.

Скопируем со страницы https://getbootstrap.com/docs/4.4/components/navs/ один из вариантов горизонтального меню, например, Working with flex utilities, и вставим этот код в обертку класса **menu** после начала области header.

Далее необходимо убрать (или добавить) гиперссылки для всех пунктов меню и отредактировать их с учетом ваших страниц и их URL.

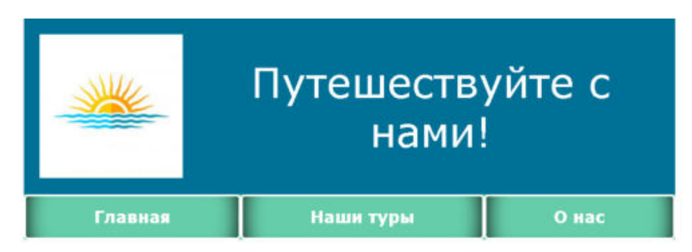
```
<!-- Обертка для меню -->
<div id = "menu">
  <!-- Начало flex-меню -->
  <nav class = "nav nav-pills flex-column flex-sm-row">
    <!-- Ссылка на главную страницу -->
    <a class = "flex-sm-fill text-sm-center nav-link" href = "index.
html">Главная</a>
    <!-- Ссылка на страницу с информацией о турах -->
    <a class = "flex-sm-fill text-sm-center nav-link" href = "tours.
html">Наши туры</a>
    <!-- Ссылка на страницу с контактами -->
    <a class = "flex-sm-fill text-sm-center nav-link" href =
"contacts.html">О нас</a>
  </nav>
</div>
```

Далее переопределим некоторые стили этого меню так, чтобы они больше походили на меню из предыдущего проекта:

```
#menu {
  /* Фон всего меню */
  background-color:#66CDAA;
}
```

```
a {
  /* Цвет и рамка каждого пункта меню */
  color:white;
  border:solid white;
  box-shadow: 20px 0 20px -20px #000 inset, -20px 0 20px -20px
#000 inset;
}
```

Получим:



Но главное, что без нашего вмешательства, за счет применения стилей и технологий Bootstrap, это меню полностью адаптивно. При уменьшении экрана получим:



Шаг 4. Создание адаптивного содержимого страниц с применением сетки Bootstrap.

Пользуясь сеткой, сверстаем контент главной страницы, который должен состоять из трех колонок, каждая из которых содержит картинку и ссылку на подробное описание. Каждая колонка занимает треть ширины экрана, а, значит, 12/3 = 4 col. Этот класс колонок и будем применять в строке (row), которая содержит основной контент страницы. Однако, так как все три колонки одинаковой ширины, вместо класса col-4 можно применять просто класс col.

```
<!-- Начало всего блока с содержимым -->
<main class = "flex">
```

```
  <!-- Начало первой строки в этом блоке -->
  <div class = "row">
    <!-- Начало единственной колонки в первой строке -->
    <div class = "col">
      Наши акции
    </div>
  </div>
  <!-- Начало второй строки в этом блоке -->
  <div class = "row">
  <!-- Начало первой колонки во второй строке -->
  <div class = "col">
    <!-- Создаем еще один блок, который объединит картинку
и ссылку в этой колонке -->
    <div class = "tour">
      <!-- Выводим картинку -->
      <img src = "img/crimea.jpg">
      <p>
        <!-- С новой строки выводим ссылку -->
        <a class = "actia" href = "#" >Подробнее об акции</a>
      </p>
    </div>
  </div>
  <!--Аналогично описываем еще две колонки в пределах второй
строки -->
  <div class = "col">
    <div class = "tour">
      <img src = "img/kavkaz.jpg">
      <p>
        <a class = "actia" href = "#" >
        Подробнее об акции</a>
      </p>
      <!--Закрываем блок с картинкой и ссылкой -->
    </div>
  <!-- Закрываем колонку -->
  </div>
    <div class = "col">
      <div class = "tour">
        <img src = "img/altay.jpg">
        <p>
          <a class = "actia" href = "#">
          Подробнее об акции</a>
        </p>
    </div>
  </div>
```

Незначительные переопределения стилей:

```
/* Для класса tour, который объединяет картинку и ссылку */
.tour {
  border: solid #007196;
}
```

```
/* Для картинок, но только входящих в блок класса tour */
.tour img {
  width: 250px;
  height: 170px;
  padding: 5px;
}
/* Для ссылок, но только входящих в блок класса tour */
.tour a {
  background-color: #66CDAA;
  box-shadow: 20px 0 20px -20px #000 inset, -20px 0 20px -20px
#000 inset;
  color: white;
}
```

Получим следующий результат для полного экрана:



И для экрана мобильного устройства:



Шаг 5. Добавление футера (нижней части каждой страницы).

Осталось только добавить footer, чтобы страница получила законченный вид.

```
<footer class = "footer" >
  <div class = "row">
    <div class = "col">
      Это сайт, предназначенный для обучения
    </div>
  </div>
</footer>
```

Стили, которые учитывают, что footer должен быть «привязан» к нижней части экрана:

```
/* Страница занимает 100% экрана по высоте*/
html {
  height: 100%;
}

/* Тело документа имеет относительное позиционирование, но отступ
для тела документа снизу больше или равен высоте футера */
body {
  min-height:100%;
  position:relative;
  padding-bottom: 70px;
}
/* Подвал документа имеет абсолютное позиционирование, высоту 50px */
.footer {
  position: absolute;
  left: 0;
  right: 0;
  bottom: 0;
  height: 50px;
  color:white;
  background-color: #007196;
}
```

Результат:



Очевидно, что теперь для создания других страниц этого сайта можно использовать страницу index.html в качестве шаблона, в котором нужно изменять только содержимое блока main.

Для создания остальных страниц сайта рекомендуем использовать готовые компоненты Bootstrap: формы, карточки, списки, кнопки и т. п., которые можно получить на странице https://getbootstrap.com/docs/4.4/components/.

Практическая работа № 4 СОЗДАНИЕ ДИНАМИЧЕСКИХ ЭЛЕМЕНТОВ НА САЙТЕ С ПРИМЕНЕНИЕМ ЯЗЫКА JAVASCRIPT

Цель: Ознакомиться с возможностями языка JavaScript для клиентской обработки web-страниц.

Инструментарий: 1) Любой редактор программного кода, например, notepad++; 2) любой современный браузер.

ЗАДАНИЯ

1. Добавить к сайту, созданному в предыдущей практической работе расчет и вывод стоимости тура.
2. Придумать и выполнить аналогичные (по технологии) действия, имеющие смысл в контексте собственного приложения.

Выполнение задания 1.

Шаг 1. Создание кнопок для вызова скрипта.

Предположим, что у вас имеется такая страница сайта, имеющая имя tour.html:



А при изменении размера экрана страница будет выглядеть так:



Код этой страницы приведен ниже (только область main).

```
<main class = "flex">
  <div class = "row">
    <div class = "col">
      Наши туры
    </div>
  </div>
  <div class = "container-fluid">
    <div class = "row">
      <div class = "col">
        <div class = "tour">
          <img src = "img/crimea.jpg" class = "tour_img">
          <p>Это прекрасная возможность отдохнуть всей семьей</p>
          <!-- Код компонента кнопки вызова модального окна
bootstrap со страницы https://getbootstrap.com/docs/4.4/
components/modal/. Значение data-target задайте свое! -->
          <button type = "button" class = "btn btn-primary"
                  data-toggle = "modal" data-target = "#myModal">
          Рассчитать стоимость
          </button>
        </div>
      </div>
    </div>
    <div class = "row">
      <div class = "col">
        <div class = "tour">
          <img src = "img/kavkaz.jpg" class = "tour_img">
          <p>Море, солнце и горы!</p>
          <button type = "button" class = "btn btn-primary"
```

```
                data-toggle = "modal" data-target = "#myModal">
            Рассчитать стоимость
          </button>
        </div>
      </div>
    </div>
    <div class = "row">
      <div class = "col">
        <div class = "tour">
          <img src = "img/altay.jpg" class = "tour_img">
          <p> Незабываемые впечатления!</p>
          <button type = "button" class = "btn btn-primary"
                data-toggle = "modal" data-target = "#myModal">
            Рассчитать стоимость
          </button>
        </div>
      </div>
    </div>
  </div>
  </div>
</main>
```

То есть на странице есть три области, каждая из которых занимает строку целиком, в которой расположены картинка, некоторый текст и кнопка для расчета стоимости соответствующего тура. Нажатие на каждую кнопку должно вызывать открытие модального окна с идентификатором myModal.

Далее необходимо добавить стили к таблице стилей:

```
.tour{
  border: solid #007196;
  margin: auto;
  height: 100%;
  /* установили возможность использования технологии flex */
  display: flex;

  /* указали направление расположения внутренних элементов –
по строке (row), и одновременно задали возможность для них перено-
ситься на другую строку при уменьшении экрана */
  flex-flow: row wrap;
}

.tour img {
  width: 250px;
  height: 170px;
  padding: 5px;
  }
.tour p .tour img {
  margin: 0 10px;
  }
```

```
/* Для того, чтобы прижать кнопку к правому краю, если элемент, 
который нужно прижать к правому краю, не последний в контейнере */
.btn {
  margin-left: auto;
  order:999;
  background-color:#66CDAA;
  box-shadow: 20px 0 20px -20px #000 inset, -20px 0 20px -20px 
#000 inset;
  color:white;
}
```

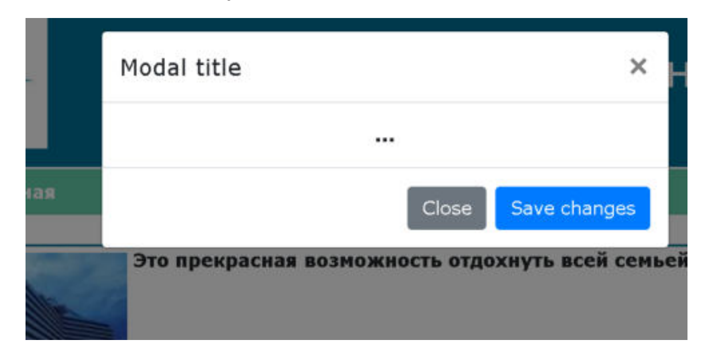
Шаг 2. Создание модального окна.

Организуем вывод модального окна, которое открывается после нажания на кнопку «Рассчитать стоимость». Это окно создадим с помощью Bootstrap. Скопируем код модального окна (https://getbootstrap.com/docs/4.4/components/modal/) в верхнюю часть кода страницы.

```
<body class = "body-top">
...
  <!-- Modal -->
<!-- Укажите id этого модульного окна таким же, как был параметр 
data-target в кнопке его вызова на предыдущем шаге -->

  <div class = "modal fade" id = "myModal" tabindex = "-1"
             role = "dialog"
             aria-labelledby = "exampleModalScrollableTitle"
             aria-hidden = "true">
    <div class = "modal-dialog modal-dialog-scrollable" role = 
"document">
      <div class = "modal-content">
        <div class = "modal-header">
        <h5 class = "modal-title"
            id = "exampleModalScrollableTitle">Modal title</h5>
        <button type = "button" class = "close"
                data-dismiss = "modal" aria-label = "Close">
          <span aria-hidden = "true">&times;</span>
        </button>
      </div>
      <div class = "modal-body">
        ...
      </div>
      <div class = "modal-footer">
        <button type = "button" class = "btn btn-secondary" data-
dismiss = "modal">Close</button>
        <button type = "button" class = "btn btn-primary">Save 
changes</button>
      </div>
    </div>
  </div>
</div>
```

В результате, при нажатии на любую из кнопок Рассчитать стоимость» должно получиться:



То есть в Bootstrap уже созданы все необходимые CSS и JS-скрипты, которые организовывают вывод этого пустого модального окна.

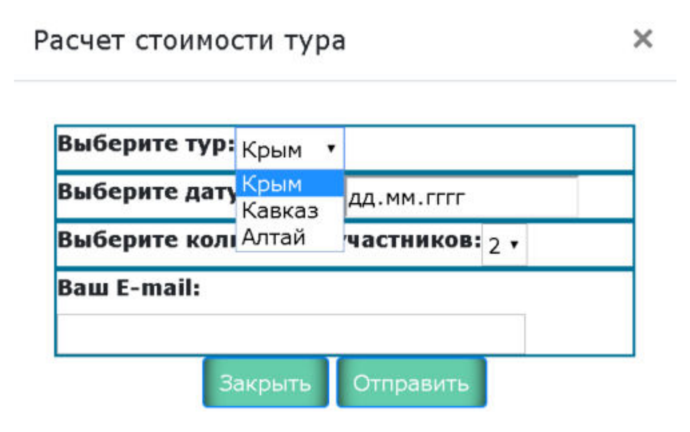
Шаг 3. Создание собственной формы в модальном окне.

Создадим в модальном окне форму для запроса у пользователя необходимых нам данных для расчета стоимости тура. Предположим, что пользователь должен выбрать название тура, дату заезда, количество участников.

```
...
<!-- Начало описания модального окна bootstrap -->
<div class = "modal-body" style = "max-width: 600px">
<!-- Начало формы -->
<form method = "get" action = "#" id = “orderForm”>
  <!-- Далее в блоках класса tour (для того, чтобы к ним были применены те же стили, что и на основной странице сайта) описываются поля формы для ввода данных. В том числе: поле для выбора страны (идентификатор, который будет использоваться для дальнейшей обработки – inp1); поле для ввода даты (тип –  date, идентификатор – inp2); поле для выбора количества участников тура (идентификатор inp3) и поле для ввода email пользователя, идентификатор inp4 -->
  <div class = "tour">
  <label class = "label" for = "name">
    Выберите тур:
  </label>
  <!-- id формы (orderForm) будет использоваться в скрипте обработки формы, рассматриваемом ниже -->
  <select id = "inp1">
    <option value = "Крым" selected>
      Крым
    </option>
    <option value = "Кавказ">
      Кавказ
    </option>
    <option value = "Алтай">
```

```
      Алтай
    </option>
  </select>
  <div>
  <div class = "tour">
    <label class = "label" for = "data">
      Выберите дату начала:
    </label>
    <input type = "date" id = "inp2">
  </div>
  <div class = "tour">
    <label class = "label" for = "number">
      Выберите количество участников:
    </label>
    <select id = "inp3">
      <option value = "1" >1</option>
      <option value = "2" selected>2</option>
      <option value = "3">3</option>
      <option value = "3">4</option>
    </select>
  </div>
  <div class = "tour">
    <label class = "label" for = "email">
    Ваш E-mail:
    </label>
    <input type = "email" id = "inp4" class = "input-xlarge" style
= "width: 350px;" required = "required">
  </div>
  <!--ОБРАТИТЕ ВНИМАНИЕ! Для упрощения обработки, перенесите
кнопки управления Закрыть и Отправить из футера модального окна
в форму. Футер модального окна  при этом можно удалить -->
  <button type = "button" class = "btn btn-primary" data-dismiss =
"modal">Закрыть</button>
  <button type = "submit"  class = "btn btn-primary" id =
"submit">Отправить</button>
</form>
```

В результате при нажатии любой из кнопок вы получите модальное окно «Расчет стоимости тура».



Шаг 4. Создание скрипта для обработки формы, заполненной пользователем в модальном окне.

Напишем скрипт на JavaScript с применением возможностей библиотеки JQuery, который позволит обработать данные, введенные пользователем в модальное окно и показать результат.

В нижней части страницы tours.html, после строк подключения библиотек нужно ввести следующий код:

```
<script>
<!-- JQuery позволяет обращаться прямо к идентификатору формы.
По событию связанному со срабатыванием кнопки submit на форме,
выполняется следующая функция -->
  $('#orderForm').on('submit', function ()
  {
    <!-- получаем название тура -->
    let tour = $("#inp1").val();
    <!-- получаем дату -->
    let date = new Date($("#inp2").val());
    <!-- получаем месяц из даты -->
    let month = date.getMonth();
    <!--получаем количество участников тура и преобразуем его
в число -->
    let kol = $("#inp3").val();
    kol = Number(kol);
    <!-- получаем email -->
    let email = $("#inp4").val();
    <!-- Выполняем некоторый условный расчет, например, если тур
в Крым, то в определенные месяцы цена выше, а в остальные – ниже
-->
    let stoim = 0;
    if (tour == 'Крым')
    {
      if((month == 5)||(month == 6)||(month == 7)||(month == 8))
        {stoim = kol*500;}
      else {stoim = kol*300;}
    }
    if (tour == 'Кавказ')
    {
      if((month == 12)||(month == 1)||(month == 2)||(month ==
5)||(month == 6)||(month == 7)||(month == 8))
       {stoim = kol*300;}
      else {stoim = kol*250;}
    }
    if (tour == 'Алтай')
    {
      {stoim = kol*1000;}
    }
```

```
    <!-- Выводим полученный результат -->
    alert (`Примерная стоимость вашего тура на ${kol} человек
составит ${stoim} у.е. Мы свяжемся с ВАМИ!!`);
  }
  );
</script>
```

В результате получим следующее:

Подтвердите действие

Примерная стоимость вашего тура на 2 человек составит 600 у.е.
Мы свяжемся с ВАМИ!!

OK

Практическая работа № 5
СОЗДАНИЕ БАЗЫ ДАННЫХ ДЛЯ САЙТА

Цель: ознакомиться с возможностями хранения данных сайта в базе данных СУБД MySQL.

Инструментарий: 1) любой редактор программного кода, например, notepad++; 2) любой современный браузер; 3) пакет разработчика Open Server (можно бесплатно скачать на сайте https://ospanel.io/download/).

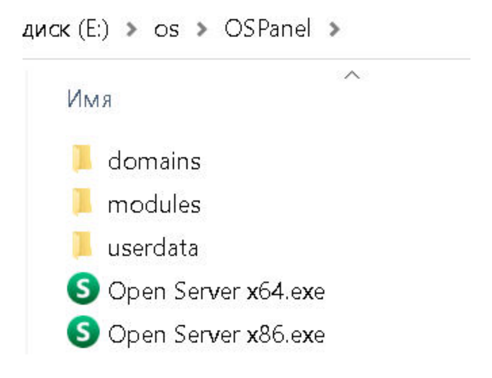
Задания

1. Развернуть локальный web-сервер и создать на нем базу данных для сайта Путешествуйте с нами. База данных должна позволять регистрироваться и авторизоваться пользователям и оставлять отзывы.
2. Придумать и выполнить аналогичные (по технологии) действия, имеющие смысл в контексте собственного приложения.

Выполнение задания 1.

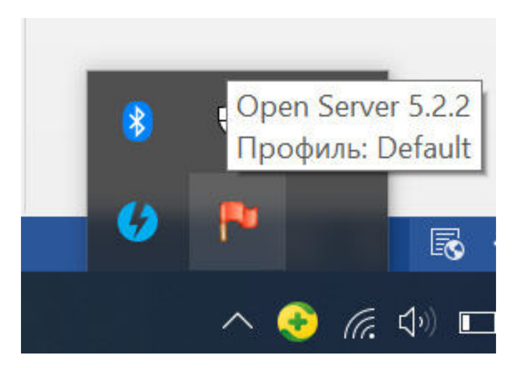
Шаг 1.Установка среды разработки.

Скачайте с сайта пакет разработчика Open Server и установите его на своем компьютере. По окончании установки вы увидите в указанной для установки папке следующую структуру:

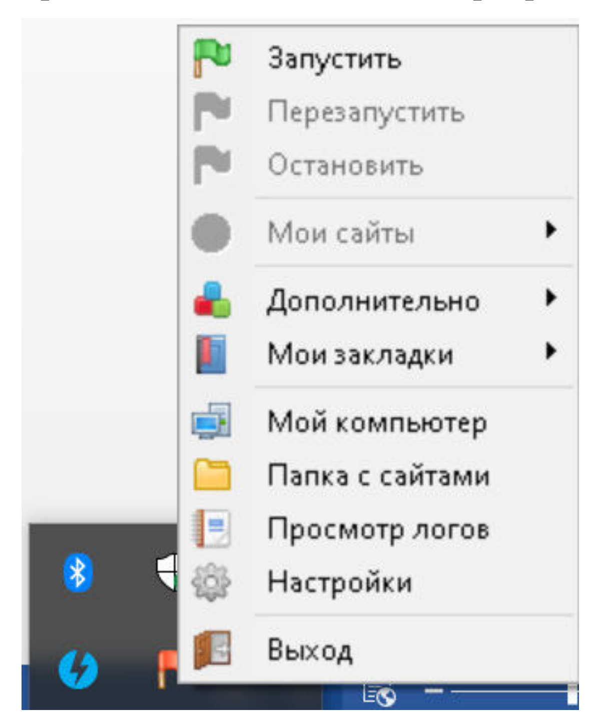


Для загрузки сервера необходимо запустить один из exe-файлов, соответствующих вашей версии операционной систе-

мы. После этого в панели задач Windows вы должны увидеть красный флажок:



Далее необходимо нажать этот флажок, что приводит к появлению меню работы с данным пакетом разработчика:



Очевидно, что для запуска сервера нужно выбрать зеленый флажок.

Теперь для проверки работы нужно в этом меню выбрать: ***Мои Сайты, localhost.*** При этом вы должны увидеть окно приветствия в браузере.

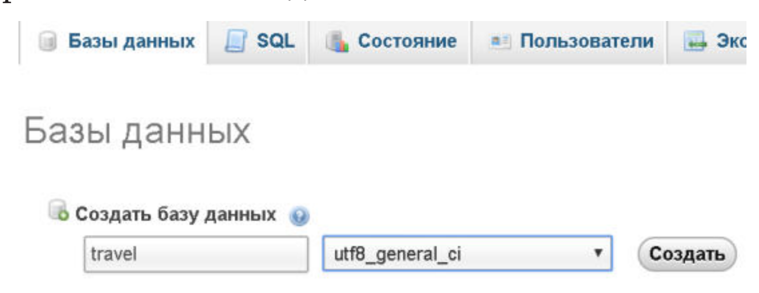
Пакет разработчика Open Server содержит программу-интерпретатор для PHP (см. тему 6) и СУБД MySQL с возможностью управлять ею через специальную программу, которая называется PHPMyAdmin. Для загрузки этой программы необходимо выбрать в меню OpenServer: ***Дополнительно, PHPMyAdmin***

Шаг 2. Создание Базы данных.

При входе система запросит логин и пароль. Как правило, логин — root, а поле пароль нужно оставить пустым.

Теперь мы попали в среду СУБД MySQL. Здесь мы будем создавать базу данных для сайта.

Выберем в меню Базы данных и заполним его так:



При нажатии на кнопку ***Создать*** вы увидите новую базу данных в левой части экрана. Дважды щелкнем по имени travel, чтобы попасть в окно редактирования этой базы данных.

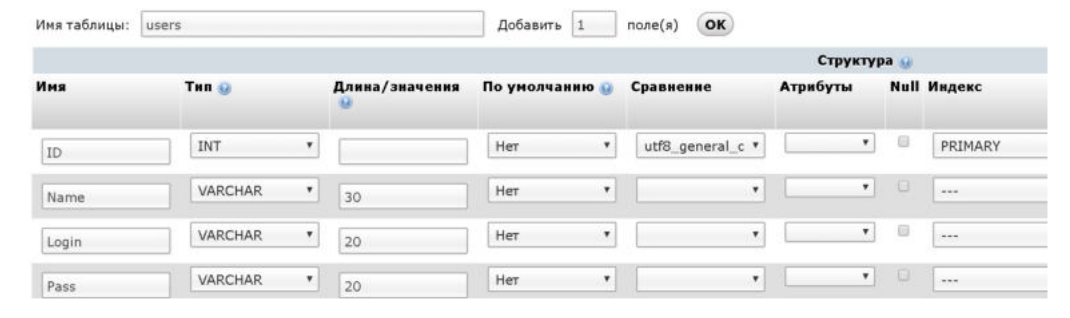
Шаг 3. Создание таблиц в базе данных.

Сначала создадим таблицу для регистрации пользователей.

Нажмем на кнопку ***Создать таблицу.***

Введем имя таблицы, например, *users*, укажем количество столбцов (4 — Порядковый номер, Имя, Логин, Пароль).

После создания, заполняем полученное окно, например, так:

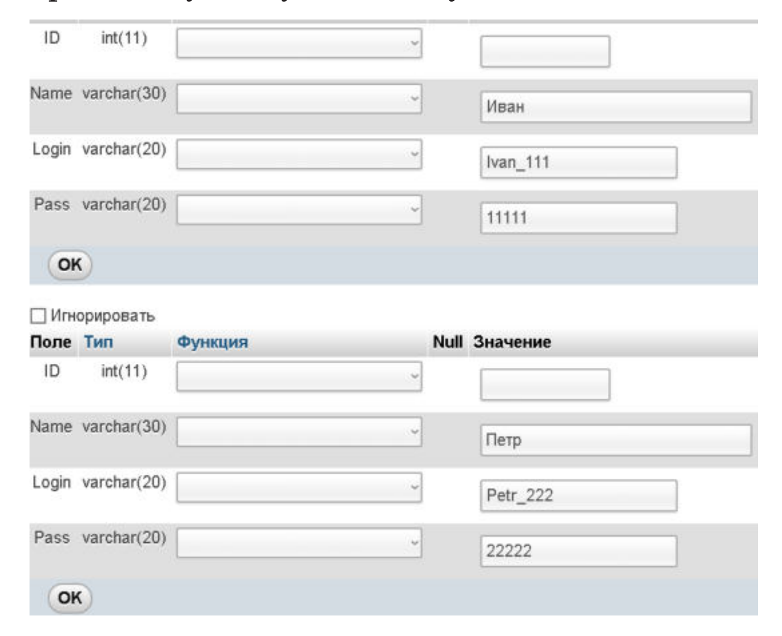


Таким образом, ID будет содержать порядковый номер зарегистрированного пользователя, это поле будет первичным ключем таблицы (PRIMARY). Далее отметим галочкой поле Auto_Increment, для того, чтобы порядковые номера заполнялись автоматически, по мере регистрации пользователей.

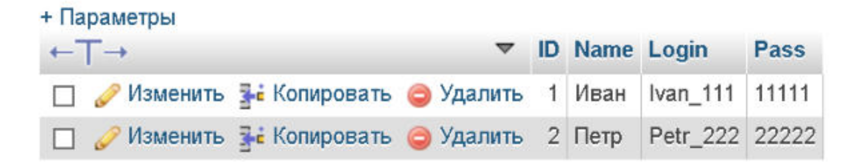
Остальные поля будут текстовыми. В поле длина устанавливается длина поля в символах (максимально необходимая для хранения данных в этом поле, например, логин не может превышать 20 символов).

После сохранения появится пустая таблица users. Если впоследствии нужно будет вносить изменения в состав или описания полей, в меню выбирается режим ***Структура***. Для заполнения этой таблицы данными выбираем режим ***Вставить***. Для просмотра — режим ***Обзор***.

Попробуйте вставить сведения о некоторых пользователях и просмотреть полученную таблицу.



После выбора опции Просмотр в меню должно получиться примерно так:



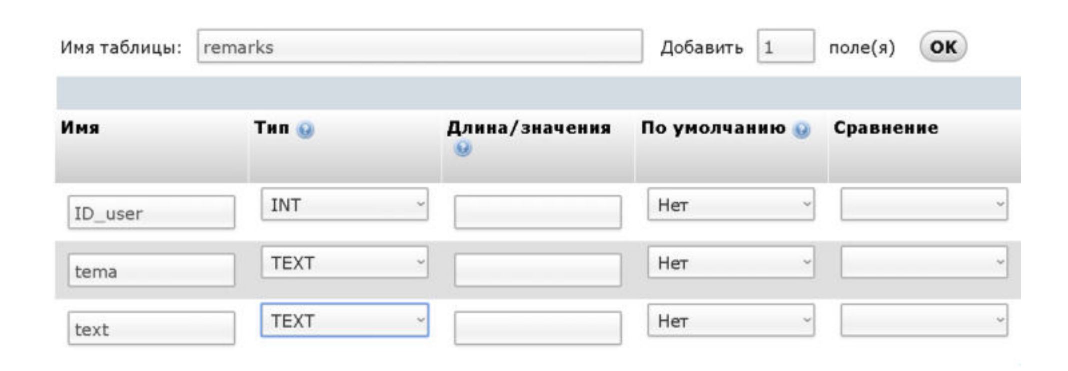
+ Параметры

	ID	Name	Login	Pass
Изменить Копировать Удалить	1	Иван	Ivan_111	11111
Изменить Копировать Удалить	2	Петр	Petr_222	22222

Вы можете заметить, что поле ID заполнилось порядковыми номерами пользователей автоматически.

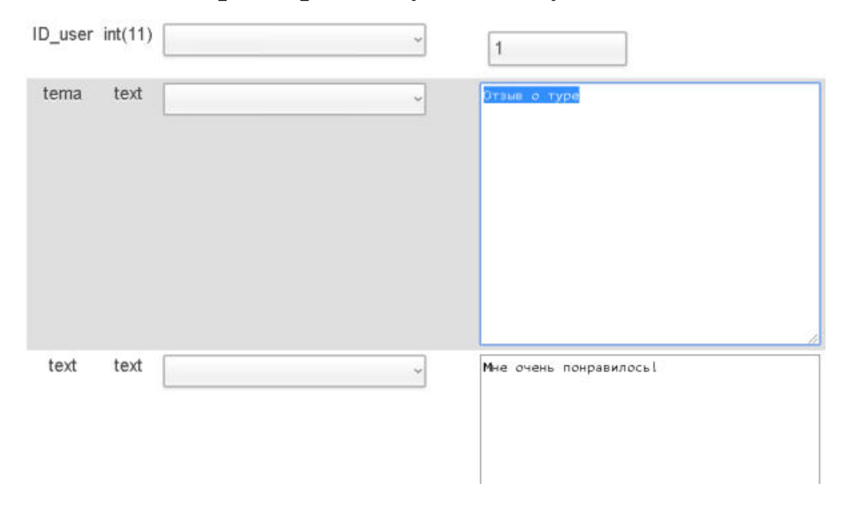
Также вы можете видеть, что пароли пользователей хранятся в базе данных в незашифрованном виде. Это, безусловно, неправильно, но решение этой проблемы не входит в задачи данного курса.

Теперь создадим еще одну таблицу для сохранения отзывов зарегистрированных пользователей. Каждый пользователь может оставлять множество отзывов. Поэтому структура второй таблицы Remarks будет следующей:



Обратите внимание: в эту таблицу входит такое же поле ID_user, как и в первую, но в этой таблице оно не является уникальным (ключевым), потому что значение этого поля может повторяться. Каждый пользователь из таблицы users может оставлять много отзывов, которые будут привязаны к его ID_user в таблице remarks. Следовательно, **не нужно** устанавливать для этого поля в этой таблице параметры PRIMARY и AUTO_INCREMENT.

Заполним для примера и эту таблицу:



Таким же образом можно было бы вставить еще множество отзывов от пользователя с ID_user 1, от пользователя с ID_user 2 и т. д.

Но наша задача — заполнять эти таблицы через сайт. Этим мы займемся в следующей практической работе.

Практическая работа № 6
РАБОТА С БАЗОЙ ДАННЫХ САЙТА С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА PHP

Цель: ознакомиться с возможностями работы с базой данных посредством запросов в PHP-скриптах.

Инструментарий: 1) любой редактор программного кода, например, notepad++; 2) любой современный браузер; 3) пакет разработчика Open Server.

Задания

1. Создать формы для регистрации, авторизации и отправки отзывов пользователей. Написать программы для регистрации и авторизации пользователей, а также для обработки функции сохранения отзывов и просмотра их на одной из страниц сайта.

2. Придумать и выполнить аналогичные (по технологии) действия, имеющие смысл в контексте собственного приложения.

Выполнение задания 1

Шаг 1. Настройка проекта в среде web-сервера.

С этого момента наш сайт будет работать под управлением локального web-сервера. Поэтому папку со всеми файлами сайта (в данном случае ***travel*** копируем в папку сервера: **[*Ваш диск*:]\[*Ваша папка сервера*]\ *OSPanel**domains***

Для запуска сайта в меню Open Server нужно выбрать ***Мои сайты, travel.***

Важно, чтобы все страницы сайта были сохранены в кодировке UTF-8 без BOM.

Шаг 2. Создание сессии.

Вы замечали, что после регистрации пользователя на сайте, в дальнейшем ему нужно только авторизоваться (то есть ввести логин и пароль) для того, чтобы система узнавала его и предоставляла конкретно этому пользователю дополнительные воз-

можности. В нашем случае — это возможность оставить отзыв. Для постоянного взаимодействия с авторизованным пользователем нужно ***использовать механизм сессий***. При авторизации пользователя сервер генерирует и запоминает уникальный ключ — идентификатор сессии, и сообщает его браузеру, затем браузер сохраняет этот ключ и при каждом последующем запросе отправляет его серверу, чтобы он «понимал» с каким пользователем происходит это взаимодействие. То есть при загрузке сайта должна стартовать новая сессия.

Поэтому теперь пересохраним файл index.html с расширением .php и сделаем его главной точкой входа на сайт. Файл index.html теперь можно удалить. Естественно, все ссылки на всех страницах, которые вели раньше на страницу index.html, должны теперь вести на страницу index.php.

В начало страницы index.php вставьте следующий код:

```
<?php
  // Запускаем сессию
  session_start();
?>
```

Шаг 3. Создание подключения к базе данных.

В папке проекта (**теперь все делаем в папке на web-сервере**) создайте файл dbconnect.php следующего содержания:

```
<?php
/* Создаем новое подключение к серверу localhost и базе данных
travel на нем. Если вы не указывали ничего дополнительно при соз-
дании вашей базы данных, то имя пользователя будет root, а пароля
не будет */
  $mysqli = new mysqli('localhost', 'root', ", 'travel');
// Проверяем, прошло ли подключение без ошибок
if ($mysqli->connect_error)
{
  die(' Connect Error ('.$mysqli->connect_errno.') '.$mysqli-
>connect_error);
}
?>
```

Это файл мы будем подключать к работе всегда, когда будет требоваться связь с базой данных.

Шаг 4. Создание области регистрации и авторизации.

На главной странице *index.php* после области заголовка должна быть реализована область, которая содержит:

1 вариант — кнопки «Регистрация» и «Авторизация»;

2 вариант — кнопки «Оставить отзыв» и «Выйти» (если пользователь авторизовался).

Поэтому добавим к коду страницы *index.php* после области с заголовком сайта следующий фрагмент:

```
<header>
...

<?php
  /* Если пользователь еще не авторизован на сайте, его переменные
сессии (мы организуем их позже) пока пусты, нужно вывести ссылки
на регистрацию или авторизацию */
  if (empty($_SESSION['login']) or empty($_SESSION['id']))
  {
?>
// скрипт прерывается и следует обычный html-код
<div id = "auth_block">
  <div id = "link_register">
    <a href = "registr.php">Регистрация</a>
  </div>
  <div id = "link_auth">
    <a href = "avtor.php">Авторизация</a>
  </div>
</div>
// скрипт снова начинается, чтобы обработать вариант else
<?php
  }
  /* Если пользователь уже авторизован, нужно выводить ссылки
на страницу, где можно оставить отзыв, или на страницу, которая
позволит выйти из авторизации */
  else
  {
?>
<div id = "exit_block">
  <div id = "link_remark">
    <a href = "remarks.php">Вы можете оставить отзыв</a>
  </div>
  <div id = "link_exit">
    <a href = "exit.php">Выход</a>
  </div>
</div>
<?php
    }
?>
```

Добавьте самостоятельно в таблицу стилей описание, которое позволит этим ссылкам быть более заметными на фоне заголовка.

Получим примерно такой результат (пока пользователь не зарегистрирован или не авторизован):

ПУТЕШЕСТВУЙ С НАМИ!

Регистрация
Авторизация

В дальнейшем обработка каждой функции будет проходить по одной и той же схеме (использованы материалы с сайта https://ruseller.com):

— ссылка на главной странице ведет на соответствующую PHP-страницу для заполнения формы;

— кнопка на этой форме запускает PHP-страницу-обработчик для этой формы;

— в конце этого обработчика располагается ссылка для возврата на одну из страниц для продолжения работы с сайтом.

Шаг 5. Реализация функции регистрации пользователя.

Создадим PHP-файл для регистрации. Согласно ссылкам, его имя должно быть *registr.php*. Создадим под этим именем новый файл следующего содержания:

```
<html>
  <head>
    <title>Регистрация</title>
  </head>
  <body>
    <h2>Регистрация</h2>
    <!-- Начало формы регистрации. save_user.php - это адрес обра-
ботчика, то есть после нажатия на кнопку "Зарегистрироваться" дан-
ные из полей будут переданы файлу save_user.php методом "post" -->
    <form action = "save_user.php" method = "post">
      <!-- В текстовое поле с именем name пользователь вводит свое
имя -->
      <p>
        <label>Ваше имя:<br></label>
        <input name = "name" type = "text" size = "15" maxlength =
"15">
      </p>
      <!-- В текстовое поле с именем login пользователь вводит
свой логин -->
      <p>
        <label>Ваш логин:<br></label>
```

```
        <input name = "login" type = "text" size = "15" maxlength
= "15">
      </p>
      <!-- В поле для паролей с именем pass пользователь вводит
свой пароль -->
      <p>
        <label>Ваш пароль:<br></label>
        <input name = "pass" type = "password" size = "15"
maxlength = "15">
      </p>
      <!-- Кнопкой (type = "submit") пользователь сможет отправить
данные на обработку файлу save_user.php -->
      <p>
        <input type = "submit" name = "submit" value = "Зареги-
стрироваться">
      </p>
    </form>
  </body>
</html>
```

При нажатии на кнопку Регистрация пользователь увидит такую страницу:

Регистрация

Ваше имя:

Ваш логин:

Ваш пароль:

Зарегистрироваться

Задание стилей этой формы или ее оформление в стиле всего сайта остается для самостоятельного выполнения.

Далее необходимо написать страницу обработки *save_user.php*.

```
<?php
  /* сохраняем введенное пользователем в форме имя в переменную
$name, если оно пустое, то уничтожаем переменную */
```

```
  if (isset($_POST['name'])) {
    $name = $_POST['name'];
    if ($name == '') {
      unset($name);
    }
  }
  /* сохраняем введенный пользователем в форме логин в переменную
$login, если он пустой, то уничтожаем переменную */
  if (isset($_POST['login'])) {
    $login = $_POST['login'];
    if ($login == '') {
      unset($login);
    }
  }
  /* сохраняем введенный пользователем пароль в форме в переменную
$pass, если он пустой, то уничтожаем переменную */
  if (isset($_POST['pass'])) {
    $pass = $_POST['pass'];
    if ($pass == '') {
      unset($pass);
    }
  }
  /* если пользователь не ввел логин или пароль, то выдаем ошибку
и останавливаем скрипт */
  if (empty($login) or empty($pass))
  {
    exit ("Вы ввели не всю информацию, вернитесь назад и заполните
все поля!");
  }
  /* подключаемся к базе данных, выполняем в этом месте ранее соз-
данный файл dbconnect.php */
  include ("dbconnect.php");

  /* Проверяем существование в базе данных пользователя с таким
же логином. Ищем в таблице users строку, где логин совпадает
с тем, который ввел пользователь, и, если находим, то сохраняем
его ID в переменной $result */
  $result = $mysqli->query("SELECT ID FROM users WHERE login =
'$login' ");

  /* Преобразуем полученный набор данных в ассоциативный массив */
  $myrow = $result->fetch_assoc();

  /* Проверяем, не пусто ли в этом ассоциативном массиве значение
поля с ключом ID. Если не пусто, то такой пользователь уже есть
в базе данных */
  if (!empty($myrow['ID'])) {
    exit ("Введенный вами логин уже зарегистрирован. Введите дру-
гой логин.");
  }
```

```
  // если такого нет, то сохраняем данные нового пользователя
в таблицу users
  $result2 = $mysqli->query ("INSERT INTO users (Name, Login,
Pass) VALUES('$name','$login','$pass')");

  // Проверяем, есть ли ошибки
  if ($result2 == 'TRUE')   {
    echo "Вы успешно зарегистрированы! Теперь вы можете зайти
на сайт под своим именем. <a href = 'index.php'>Главная страница</
a>";
  }
  else {
    echo "Ошибка! Вы не зарегистрированы.";
  }
?>
```

Перейдите на форму регистрации, заполните ее и убедитесь в том, что новая строка появилась в базе данных в таблице *users*.

Шаг 6. Реализация функции авторизации.

После успешной регистрации пользователь может перейти с главной страницы по ссылке Авторизация. При этом он должен ввести свой логин и пароль, и если в таблице users будет строка с такими же логином и паролем, то пользователю должна стать доступна какая-либо дополнительная функция сайта. В нашем случае при переходе на главную страницу он должен увидеть ссылки «Оставить отзыв» и «Выйти».

Создаем форму авторизации (*avtor.php*):

```
<html>
  <head>
    <title>Авторизация</title>
  </head>
  <body>
    <h2>Авторизация</h2>
    <!-- Начало формы авторизации. test_user.php - это адрес обра-
ботчика, то есть после нажатия на кнопку “Войти” данные из полей
передаются обработчику test_user.php методом “post” -->
    <form action = "test_user.php" method = "post">
      <!-- В текстовое поле  с именем login пользователь вводит
свой логин -->
      <p>
        <label>Ваш логин:<br></label>
        <input name = "login" type = "text" size = "15" maxlength
= "15">
      </p>
      <!-- В поле для паролей с именем pass пользователь вводит
свой пароль -->
```

```
      <p>
        <label>Ваш пароль:<br></label>
        <input name = "pass" type = "password" size = "15" 
maxlength = "15">
      </p>
      <!-- Кнопка (type = "submit") отправляет данные обработчику 
test_user.php -->
      <p>
        <input type = "submit" name = "submit" value = "Войти">
      </p>
    </form>
  </body>
</html>
```

Результат:

Авторизация

Ваш логин:

Ваш пароль:

Войти

Создаем файл-обработчик (*test_user.php*):

```
<?php

  // Обязательно запускаем сессию!
  session_start();
  // Сохраняем введенные пользователем логин и пароль в переменные 
$login и $pass
  if (isset($_POST['login'])) {
    $login = $_POST['login'];
    if ($login == ") {
      unset($login);
    }
  }
  if (isset($_POST['pass'])) {
    $pass = $_POST['pass'];
    if ($pass == ") {
      unset($pass);
    }
  }
  // Если пользователь не ввел логин или пароль, то выдаем ошибку 
и останавливаем скрипт
```

```
  if (empty($login) or empty($pass)) {
    exit ("Вы ввели не всю информацию, вернитесь назад и заполните 
все поля!");
  }
  // Подключаемся к базе
  include ("dbconnect.php");
  // Извлекаем из базы все данные о пользователе с введенным логи-
ном
  $result = $mysqli->query("SELECT * FROM users WHERE login = 
'$login'");
  // Помещаем эти данные в ассоциативный массив
  $myrow = $result->fetch_assoc();
  // Проверяем, существует ли пользователь с таким логином
  if (empty($myrow['Login'])) {
    exit ("Введенный вами login или пароль неверный.");
  }
  else {
    // Если пользователь в базе существует, то сверяем пароли
    if ($myrow['Pass'] == $pass) {
      /* Если пароли совпадают, то запускаем пользователю сессию. Это 
означает, что в специальных сессионных переменных сохраняются важные 
сведения об этом пользователе, в данном случае – ID и логин */
      $_SESSION['login'] = $myrow['Login'];
      $_SESSION['id'] = $myrow['ID'];
      // Выводятся соответствующие сообщения пользователю
      echo "Вы успешно вошли на сайт! <a href = 'index.
php'>Главная страница</a>";
    }
    else {
      // Если пароли не сошлись
      exit ("Введенный вами login или пароль неверный.");
    }
  }
?>
```

Перейдите по ссылке Авторизация и введите один из логинов и паролей зарегистрированных ранее пользователей. Убедитесь, что функция работает.

Убедитесь в том, что теперь после перехода на главную страницу ссылки «Авторизация» и «Регистрация» заменены на ссылки «Вы можете оставить отзыв» и «Выйти». Это произошло потому, что переменные сессии теперь не пусты, и это условие сработало при загрузке страницы.

Шаг 7. Реализация функции «Оставить отзыв».

Создадим форму для отзыва (файл *remarks.php*)

```
<html>
  <head>
    <title>Отзывы</title>
  </head>
```

```
  <body>
    <h2>Оставьте свой отзыв</h2>
    // Форма будет обработана файлом save_remarks.php
    <form action = "save_remarks.php" method = "post">
      <!-- В текстовую область (name = “tema”) пользователь вводит
тему своего отзыва -->
      <p>
        <label>Тема сообщения:<br></label>
        <textarea name = "tema" cols = "38" rows = "3"></textarea>
      </p>
      <!-- В текстовую область (name = “text”) пользователь вводит
текст своего отзыва -->
      <p>
        <label>Введите текст сообщения:<br></label>
        <textarea name = "text" cols = "38" rows = "6"></textarea>
      </p>
      <!-- Кнопка отправляет данные на страницу-обработчик save_
remarks.php -->
      <p>
        <input type = "submit" name = "submit" value = "Сохра-
нить">
      </p>
    </form>
```

И, наконец, файл-обработчик *save_remarks.php*. Этот файл должен сохранить введенный отзыв в таблице *remarks* нашей базы данных и перенаправить пользователя на страницу (например *contacts.php*), где будут выведены все имеющиеся в базе данных отзывы.

```
<?php
// Старт сессии
session_start();
/* Получаем из формы и сохраняем в соответствующих переменных вве-
денные тему и текст отзыва */
if (isset($_POST['tema'])) {
  $tema = $_POST['tema'];
  if ($tema == '') {
    unset($tema);
  }
}
if (isset($_POST['text'])) {
  $text = $_POST['text'];
  if ($text == '') {
    unset($text);
  }
}
// Если пользователь не ввел тему или текст, то выдаем ошибку
и останавливаем скрипт
```

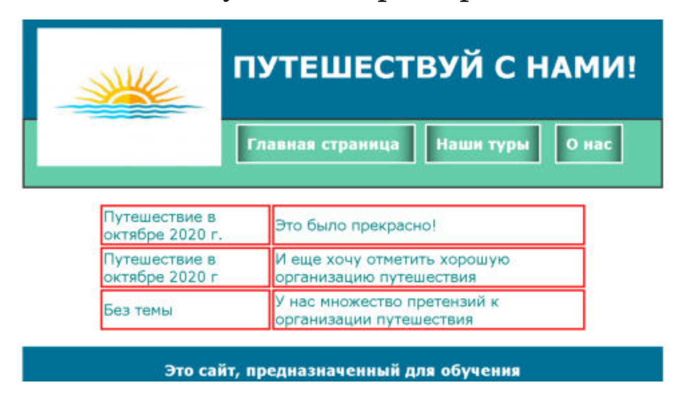
```
if (empty($tema) or empty($text)) {
  exit("Вы ввели не всю информацию, вернитесь назад и заполните
все поля!");
} else {
  // Если тема и текст введены, то сохраняем сообщение в таблицу
remarks БД travel
  // Подключаемся к базе
  include("dbconnect.php");
  // Если не пуста переменная сессии, сохраняем ID текущего поль-
зрвателя
  if (!empty($_SESSION['id'])) {
    $kod    = $_SESSION['id'];
    // Сохраняем данные
    $result = $mysqli->query("INSERT INTO remarks (ID_user, tema,
text) VALUES('$kod','$tema','$text')");
    // Проверяем, нет ли ошибки
    if ($result == 'TRUE') {
      // Перенаправляем пользователя на страницу просмотра отзывов
      echo "Ваше сообщение сохранено! Перейти к просмотру сообщений.
      <a href = 'contacts.php'>О нас</a>";
    } else {
      echo "Ошибка! Сообщение не сохранено";
    }
  }
}
?>
```

Страница просмотра всех отзывов, которые имеются в БД, может быть организована на одной из страниц сайта. Например, если нужно просматривать ее по ссылке в главном меню сайта «О нас», то в области контента страницы *contacts.php* должен быть размещен следующий код (как только мы вставляем в какую либо страницу фрагмент PHP, страница должна быть переименована с расширением .php):

```
<?php
  // Подключаем базу данных
  include("dbconnect.php");
  // Получаем все строки, которые есть в таблице remarks
  $result = $mysqli->query("SELECT * FROM remarks");
  // Начинаем строить таблицу, присваиваем ей имя $table
  $table = "<table>";
  // Начинаем вести счетчик отзывов (строк в таблице)
  $k = 1;
  // Начинаем цикл, позволяющий вывести все отзывы из таблицы
remarks
  while ($myrow = $result->fetch_assoc()) {
    $table .= "<tr>";
    $table .= "<td>".$myrow['tema']."</td>";
```

```
    $table .= "<td>".$myrow['text']."</td>";
    $table .= "</tr>";
    $k++;
  }
  // Закрываем таблицу
  $table .= "</table>";
  // Выводим сформированную таблицу
  echo $table;
?>
```

Результат должен получиться примерно таким:



Шаг 7. Выход из сессии.

Далее нам нужно понять, как обработать ссылку «Выйти» на главной странице сайта. Она должна прервать сессию данного пользователя и вновь показывать варианты Регистрация и Авторизация. Нам необходимо очистить переменные сессии и перенаправить пользователя на главную страницу, на которой после проверки будет выбран вариант загрузки, соответствующий неавторизованному доступу.

Для этого создадим еще один файл *exit.php*:

```
<?php
  session_start();
  session_unset();
  session_destroy();
  header("Location: index.php");
?>
```

Практическая работа № 7
ГЕНЕРАЦИЯ ДИНАМИЧЕСКИХ СТРАНИЦ САЙТА

Цель: закрепить на практике понимание принципов создания динамических web-приложений.

Инструментарий: 1) любой современный браузер; 2) любой редактор программного кода; 3) пакет разработчика Open Server.

Задания

1. Изменить созданный в предыдущих заданиях сайт таким образом, чтобы при помещении новой информации в БД о новых турах на главной странице автоматически появлялась динамическая галерея, а при щелчке по кнопке «Подробнее о туре» открывалась динамически сгенерированная новая страница с подробными сведениями об этом туре.
2. Выполнить подобные (по использованным технологиям) действия для своего сайта.

Шаг.1. Шаблонизация страниц.

Чтобы можно было генерировать страницы динамически, на основании информации, получаемой из базы данных, необходимо разбить каждую страницу физически на отдельные компоненты (шапку, меню, контентную часть, подвал). Таким образом можно достичь двух целей.

1) размер каждой страницы уменьшается, так как можно хранить шапку, меню и подвал сайта в специальных файлах шаблона, которые подключаются к каждой странице;

2) изменяется только контентная часть каждой страницы.

Скопируем и разместим в спецальной папке tpl шапку сайта под именем *header.php*:

```
<!DOCTYPE html>
<head>
  <meta charset = "utf-8">
```

```
  <meta name = "viewport" content = "width = device-width,
initial-scale = 1, shrink-to-fit = no">
  <!-- Стиль bootstrap -->
  <link rel = "stylesheet" href = "css/bootstrap.min.css" >
  <!-- Собственный стиль -->
  <link rel = "stylesheet" href = "css/style.css" >
  <title>Сайт туристической компании</title>
</head>
<body class = "body-top">
  <div id = "content">
  <header class = "container-fluid ">
    <div class = "row">
      <div class = "col-3">
        <img id = "logo" src = "img/logo.jpg">
      </div>
      <div class = "col-9">
        <h1>Путешествуйте с нами!</h1>
      </div>
    </div>
    <?php
      // Проверяем, пусты ли переменные логина и id пользователя
      if (empty($_SESSION['login']) or empty($_SESSION['id']))
      {
      ?>
    <div class = "row">
      <div class = "col">
        <div id = "auth_block">
          <div id = "link_register">
            <a href = "registr.php">Регистрация</a>
          </div>
          <div id = "link_auth">
            <a href = "avtor.php">Авторизация</a>
          </div>
        </div>
      </div>
    </div>
    <?php
      }
      else
      {
    ?>
    <div class = "row">
      <div class = "col">
        <div id = "exit_block">
          <div id = "link_remark">
            <a href = "remarks.php">Вы можете оставить отзыв</a>
          </div>
          <div id = "link_exit">
            <a href = "exit.php">Выход</a>
          </div>
        </div>
```

```
        </div>
      </div>
      <?php
        }
      ?>
    </header>
```

Аналогично, сохраним в отдельных файлах *nav.php* и *footer.php* в папке tpl код для меню и подвала каждой страницы.

После этого файл *index.php* можно переписать так:

```
<?php
  // Запускаем сессию
  session_start();
  // Подключаем файлы для шапки и меню
  include ('tpl/header.php');
  include ('tpl/nav.php');
?>
<! -- Здесь будет описание содержимого главной страницы -- >
<main class = "flex">

</main>
<?php
//подключаем файл для подвала
  include ('tpl/footer.php');
?>
```

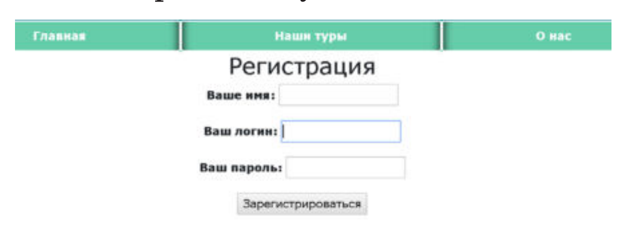
Убедитесь в том, что все компоненты шаблона подключились, а главная страница сайта выглядит также, как и до шаблонизации.

Теперь нужно так же изменить и все остальные страницы сайта — оставить на каждой странице только основное содержимое в области main, а выше и ниже этой области подключить внешние шаблоны шапки, меню и подвала. Так, например, страница *registr.php* теперь может быть переписана так:

```
<?php
  include ('tpl/header.php');
  include ('tpl/nav.php');
?>
  <main class = "flex">
  <h2>Регистрация</h2>
    <form action = "save_user.php" method = "post">
      <p>
        <label>Ваше имя:<br></label>
        <input name = "name" type = "text" size = "15" maxlength =
"15">
      </p>
```

```
      <p>
        <label>Ваш логин:<br></label>
        <input name = "login" type = "text" size = "15" maxlength
= "15">
      </p>
      <p>
        <label>Ваш пароль:<br></label>
        <input name = "pass" type = "password" size = "15"
maxlength = "15">
      </p>
      <p>
        <input type = "submit" name = "submit" value = "Зареги-
стрироваться">
      </p>
    </form>
  </main>
<?php
  include ('tpl/footer.php');
?>
```

В результате эта страница будет выглядеть так (фрагмент):



Шаг 2. Создание таблицы базы данных для хранения информации о турах.

В созданной ранее базе данных средствами PHPMyAdmin создадим новую таблицу следующей структуры:

Имя таблицы: tours Добавить 1 поле(я) OK

Имя	Тип	Длина/значения	По умолчанию	Сравнение	Атрибуты	Null	Индекс
id	INT		Нет				PRIMARY
name	VARCHAR	60	Нет				---
programm	TEXT		Нет				---
photo	VARCHAR	60	Нет				---

Таким образом, в этой таблице можно будет сохранять идентификатор тура, его наименование, его программу и одно изображение.

Добавим в таблицу несколько туров так, чтобы сведения о них могли заменить те, которые сейчас расположены статич-

но на главной странице сайта, и дополнительно еще один тур, например, в Санкт-Петербург.

В результате заполненная таблица будет выглядеть примерно так:

id	name	programm	photo
1	Крым	Приглашаем вас в туры по горному Крыму в период с ...	img/crimea.jpg
2	Кавказ	Лучшие морские курорты Кавказа ждут вас	img/kavkaz.jpg
3	Алтай	Приглашаем Вас в тур по предгорьям Алтайских гор н...	img/altay.jpg
4	Санкт-Петербург	Незабываемые впечатления от дворцов и парков север...	img/peter.jpg

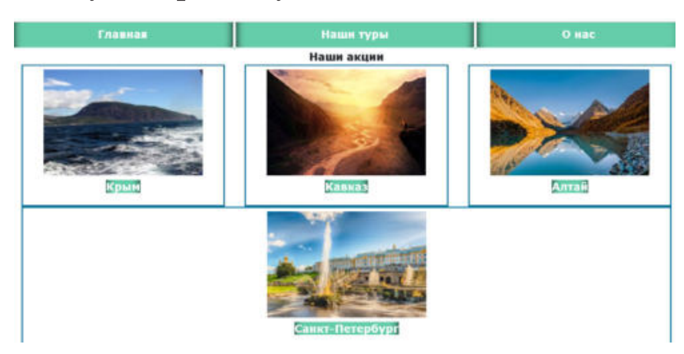
Шаг 3. Создание динамической галереи.

Выведем на главной странице вместо прежней статической галереи динамическую, то есть содержащую сведения о тех турах, которые есть в таблице *tours* БД *travel*.

Ниже описана только область main страницы *index.php*:

```
<main class = "flex">
  <div class = "row">
    <div class = "col">
      Наши новые туры
    </div>
  </div>
  <div class = "container-fluid">
    <?php
      // Выполняем подключение к БД
      // Файл dbconnect.php должен быть в той же папке
      include ("dbconnect.php");
      // Получаем все данные, имеющиеся в таблице tours
      $result = $mysqli->query("SELECT * FROM tours");
      /* Формируем область в переменной $div, в которую в цикле
помещаем все имеющиеся в базе туры (их фото и наименование в виде
ссылки) */
      $div = '<div class = "row">';
      $k = 1;
      while($myrow = $result->fetch_assoc())
      {
        $div .= '<div class = "col">
          <div class = "country">';
        $div .= '<img src = '.$myrow[photo].'>';
        $div .= '<p><a href = >'.$myrow['name'].'</a></p>';
        $div .= '</div></div>';
        $k++;
      }
      $div .= '</div>';
      // Выводим на экран содержимое переменной $div
      echo $div;
    ?>
  </div>
```

Получим такую страницу *index.html*:



Проверьте, как изменится содержимое страницы, если к таблице *tours* средствами PHPMyAdmin добавить еще один новый тур.

Шаг 4. Создание динамических страниц о любом из туров.

Далее нам необходимо понять, как вывести страницу, содержащую более подробные сведения о любом туре, при нажатии на соответствующую ссылку на главной странице. Понятно, что структура этой страницы одинакова для всех туров, однако в каждом случае должна выводится информация только из соответствуюшей строки таблицы. Поэтому при нажатии на ссылку под каждой фотографией на главной странице должна открываться страница *tour.php*, но при этом в ее контентную часть должны выводиться данные только из одной строки таблицы *tours*, id которой воответствует выбранному туру. Это может быть выполнено за счет передачи id в URL страницы *tour.php*.

Внесем следующие изменения в код страницы index.php:

```
<main class = "flex">
  <div class = "row">
    <div class = "col">
      Наши акции
    </div>
  </div>
  <div class = "container-fluid">
    <?php
      include ("dbconnect.php");
      $result = $mysqli->query("SELECT * FROM tours");
      $div = '<div class = "row">';
      $k = 1;
      while($myrow = $result->fetch_assoc())
      {
        $div . = '<div class = "col">
          <div class = "country">';
```

```
        /* Здесь мы сохраняем в переменной $id значение каждой
из перебираемых в цикле строк тура из таблицы tours базы данных */
        $id = $myrow[id];
        /* Далее при формировании ссылки этот id встраива-
ется в адрес, на который указывает ссылка. Обращайте внимание
на использование кавычек! */
        $div .= '<img src = '.$myrow[photo].'>';
        $div .= "<p><a href = tour.php?id = $id>".$myrow[name].'</
a></p>';
        $div .= '</div></div>';
        $k++;
      }
      $div . = '</div>';
      echo $div;
    ?>
  </div>
</main>
```

Таким образом, при переходе по любой из ссылок под фотографией тура мы получим переход на страницу *tour.php*, но с параметром, совпадающим с номером этого тура в таблице БД. Например, при выборе второго тура в адресной строке браузера должно появиться:

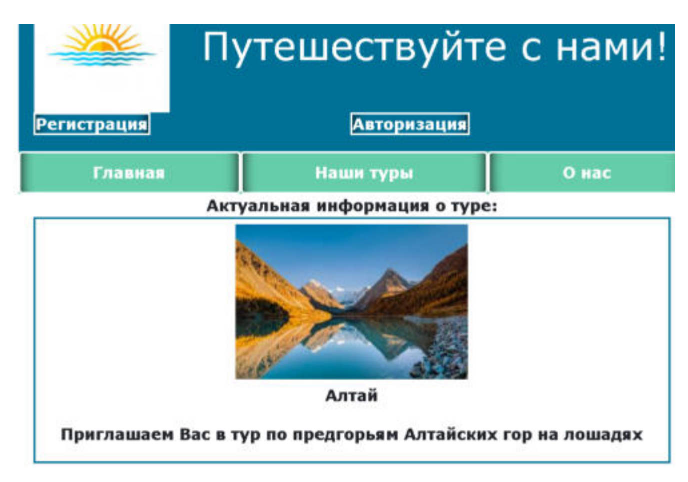
Не защищено | travel/tour.php?id=2

Этот способ позволяет также передать параметр странице *tour.php* и использовать его для поиска нужной информации в БД для последующего отображения ее на странице. Параметр передается из URL на страницу методом GET. Ниже представлен код страницы *tour.php*.

```
<?php
  include('tpl/header.php');
  include('tpl/nav.php');
?>
<main class = "flex">
  <div class = "row">
    <div class = "col">
      Актуальная информация о туре:
    </div>
  </div>
  <div class = "container-fluid">
    <?php
      // Подключаемся к базе данных
      include("dbconnect.php");
      /* Получаем из URL страницы id именно того тура, информацию
```

```
о котором нужно разместить на этой странице сейчас */
      $label = ‘id’;
      $id = false;
      if (!empty($_GET[$label])) {
        $id = $_GET[$label];
      }
      /* Выполняем запрос к таблице tours БД, чтобы получить все
сведения о туре с данным id */
      $result = $mysqli->query("SELECT * FROM tours WHERE id =
'$id'");
      /* Преобразуем строку, полученную в результате запроса
в ассоциативный массив, чтобы можно было обращаться к каждому зна-
чению по имени поля в таблице */
      $myrow = $result->fetch_assoc();
      /* Формируем переменную, содержащую структуру выводимых зна-
чений */
      $div = '<div class = "row">';
      $div .= '<div class = "col">
        <div class = "country">';
      $id = $myrow[id];
      $div .= '<img src = '.$myrow[photo].'>';
      $div .= '<p>'.$myrow[name].'</p>';
      $div .= '<p>'.$myrow[programm].'</p>';
      $div .= '</div></div>';
      $div .= '</div>';
      // Выводим содержимое этой переменной
      echo $div;
    ?>
  </div>
</main>
<?php
  include('tpl/footer.php');
?>
```

В результате получим полную информацию именно о том туре, который был выбран на странице index.php.



Практическая работа № 8 ИСПОЛЬЗОВАНИЕ REACT ДЛЯ ПРЕДСТАВЛЕНИЯ ИНФОРМАЦИИ, ПОЛУЧЕННОЙ С СЕРВЕРА

Цель: опробовать на практике современные принципы рендеринга web-страниц.

Инструментарий: 1) любой современный браузер; 2) редактор программного кода; 3) пакет разработчика Open Server.

Задания

1. Переверстать страницу index.php, разработанную в ходе выполнения лабораторной работы № 8, так, чтобы web-страница с данными из БД формировалась не на сервере с использованием PHP, а в браузере с применением фреймворка JS React.
2. Выполнить аналогичные действия для своего сайта

Выполнение задания 1

В предыдущей практической работе мы формировали динамическую страницу с карточками туров на сервере средствами PHP и передавали уже готовый HTML-код с сервера для отображения в браузере. Сейчас мы выполним «отрисовку» страницы на языке JavaScript в самом браузере, используя данные текстового файла JSON, который будет передаваться с сервера.

Шаг 1. Создание PHP-сервиса для получения результата запроса к базе данных в формате JSON.

Создадим в папке проекта следующий файл с именем *api.php*:

```
<?PHP
  // Подключаем файл соединения с базой данных
  include ("dbconnect.php");
  // Выполняем нужный запрос к базе данных
  // Извлекаем из базы все данные об имеющихся турах
  $result = $mysqli->query("SELECT * FROM tours");
  // Создаем новый пустой массив
```

```
  $tours = [];
  // Записываем все туры из таблицы БД в строки этого массива
  while($myrow = $result->fetch_assoc()) {
    $tours[] = $myrow;
  }
  // Сохраняем данные полученного массива в текстовый файл формата
JSON
  header('Content-type: application/json');
  echo json_encode($tours);
?>
```

При попытке запустить этот файл через локальный web-сервер вы увидите текстовый файл, в котором можно распознать структуру таблицы tours БД.

Шаг 2. Изменение файла index.php

Изменим файл index.php следующим образом:

```
<?php
  // Запускаем сессию
  session_start();
  /* Подключаем измененный файл header, так как теперь он должен
содержать ссылки на подключаемую библиотеку React*/
    include (‘tpl/header.php’);
    include ('tpl/nav.php');
?>
<main class = "flex">
  <div class = "row">
    <div class = "col">
      Актуальные направления путешествий
    </div>
  </div>
  <div class = "container-fluid">
  <!-- Область отрисовки динамического содержимого страницы просто
описана контейнером с id = root -->
    <div id = "root">
    </div>
  </div>
</main>
<?php
  // Подключение нового файла footer.php
  include (‘tpl/footer.php’);
?>
```

Шаг 3. Изменения в файле header.php.

```
<!DOCTYPE html>
<head>
  <!-- Required meta tags -->
  <meta charset = "utf-8">
```

```
  <meta name = "viewport" content = "width = device-width,
initial-scale = 1, shrink-to-fit = no">
  <!-- Стиль bootstrap -->
  <link rel = "stylesheet" href = "css/bootstrap.min.css" >
  <!-- Собственный стиль -->
  <link rel = "stylesheet" href = "css/style.css" >
  <!-- Добавление внешних ссылок на файлы из библиотеки React.
Взяты с сайта React -->
  <script src = "https://unpkg.com/react@16/umd/react.development.
js"></script>
  <script src = "https://unpkg.com/react-dom@16/umd/react-dom.
development.js"></script>
  <script src = "https://unpkg.com/babel-standalone@6.15.0/babel.
min.js"></script>
  <title>Сайт туристической компании</title>
</head>
<body class = "body-top">
  <div id = "content">
  <header class = "container-fluid ">
    <div class = "row">
      <div class = "col-3">
        <img id = "logo" src = "img/logo.jpg">
      </div>
      <div class = "col-9">
        <h1>Путешествуйте с нами!</h1>
      </div>
    </div>
    <?php
      // Проверяем, пусты ли переменные логина и id пользователя
      if (empty($_SESSION['login']) or empty($_SESSION['id'])) {
    ?>
    <div class = "row">
      <div class = "col">
        <div id = "auth_block">
          <div id = "link_register">
            <a href = "registr.php">Регистрация</a>
          </div>
          <div id = "link_auth">
            <a href = "avtor.php">Авторизация</a>
          </div>
        </div>
      </div>
    </div>
    <?php
      }
      else
      {
      ?>
    <div class = "row">
      <div class = "col">
        <div id = "exit_block">
```

```
          <div id = "link_remark">
            <a href = "remarks.php">Вы можете оставить отзыв</a>
          </div>
          <div id = "link_exit">
            <a href = "exit.php">Выход</a>
          </div>
        </div>
      </div>
    </div>
    <?php
      }
      ?>
  </header>
```

Шаг 4. Изменение файла footer.php.

В файл подвала необходимо внести изменения так, чтобы он вызывал скриптовый файл *my.js*, который и будет выполнять основную функцию: получать данные из файла JSON и визуализировать их на странице *index.php*.

```
    <footer class = "footer" >
      <div class = "row">
        <div class = "col">
          Это сайт, предназначенный для обучения
        </div>
      </div>
    </footer>
  </div>
  <!-- Подключение внешних библиотек JS -->
  <!-- Строки взяты со стартовой страницы bootstrap и React -->
  <script src = "https://code.jquery.com/jquery-3.4.1.slim.min.js"
integrity = "sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGF
AV0wwj1yYfoRSJoZ+n" crossorigin = "anonymous"></script>
  <script src = "https://cdn.jsdelivr.net/npm/popper.js@1.16.0/
dist/umd/popper.min.js" integrity = "sha384-Q6E9RHvbIyZFJoft+2
mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin =
"anonymous"></script>
  <script src = "https://stackpath.bootstrapcdn.com/
bootstrap/4.4.1/js/bootstrap.min.js" integrity = "sha384-wfSD
F2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6"
crossorigin = "anonymous"></script>

  <!-- Подключение собственного JS-файла -->
  <!--тип text/babel указывается для совместимости с браузерами
более ранних версий -->
  <script type = "text/babel" src = "js/my.js"></script>
</body>
</html>
```

Шаг 5. Описание файла my.js.

Основной файл при применении данной технологии рендеринга — это файл JS, который содержит код для получения внешних данных в формате JSON и отображения их на странице с применением библиотеки React. Теоретические сведения об этой библиотеке приведены в п. 5.9. первой части данного пособия.

Напоминаем, что структура таблицы с турами, данные из которой должны динамически отображаться на этой странице выглядит так:

id	name	programm	photo
1	Крым	Приглашаем вас в туры по горному Крыму в период с ...	img/crimea.jpg
2	Кавказ	Лучшие морские курорты Кавказа ждут вас	img/kavkaz.jpg
3	Алтай	Приглашаем Вас в тур по предгорьям Алтайских гор н...	img/altay.jpg
4	Санкт-Петербург	Незабываемые впечатления от дворцов и парков север...	img/peter.jpg

Файл *my.js* с комментариями приведен в следующем листинге.

```
/* Описываем функцию, которая "умеет" представлять на странице
отдельные элементы компонента React с использованием блочной вер-
стки. Компонент в данном случае представляет собой карточку тура,
включающую фото, и наименование тура. Props - это параметр, кото-
рый позволяет обрабатывать внешние для компонента данные */
function Tour(props) {
  return (
    <div className = "card">
      <div className = "card-img">
       /* выводим фото тура по имени соответствующего поля
в таблице БД */
        <img className = "tour-img" src = {props.tour.photo} />
        <h3>
            /* выводим название тура как гиперссылку на динамиче-
скую страницу */
            /* параметром в URL вызываемой страницы выступает
id тура */
            <a className = "tour-ref"
                 href = {"tour.php?id = "+props.tour.id}>{props.
tour.name}
            </a>
        </h3>
      </div>
    </div>
  )
}

/* Описываем класс компонента App (приложение), наследуемого от ком-
понента React в котором имеется его состояние (state) – в данном
случае набор всех возможных значений и рендеринг (render) – отри-
совка на странице шаблона со связанным с ним состоянием */
```

```
class App extends React.Component
{
  /*объявляем объект, который описывает состояние компонента
state*/
  state = {tours:[]}

  /* Создаем объект, который асинхронно может запросить у сервера
данные через созданный нами ранее файл api.php и распознать эти
данные как JSON. Здесь для этого используется современный интер-
фейс JS, который называется Fetch API */
  fetchQuotes = () = > {
    this.setState({...this.state, isFetching: true})
    fetch("api.php")
      .then(response = > response.json())
      /* Здесь полученные данные связываются с объектом, описанным
выше как состояние компонента */
      .then(result = > this.setState({tours: result, isFetching:
false}))
      .catch(e = > console.log(e));
  }

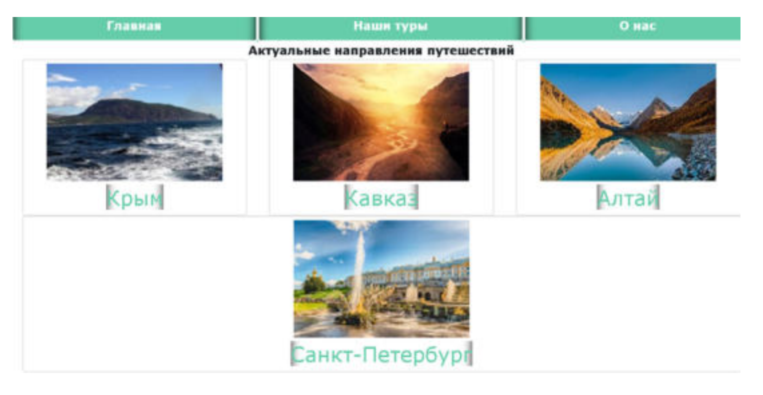
  /* У компонентов React есть методы жизненного цикла,
которые позволяют сохранять актуальное состояние. Метод
componentDidMount() вызывается, когда компонент становится доступ-
ным. Поэтому здесь мы непосредственно получаем данные */
  componentDidMount() {
    this.fetchQuotes()
  }

  /* Здесь мы указываем как на странице будет выполняться отри-
совка (рендеринг) компонента */
  render() {
    console.log(this.state)
    return(
      /* Создается обертка, которая может быть использована затем
в CSS */
      <div className = "app">
        <div className = "list">
          <div class = "row">
          {
            /* Выполняется отображение каждого текущего состояния
компонента с использованием ранее описанной функции представления
Tours */
            this.state.tours.map(tour => {
              return (
                <div class = "col">
                  <Tour tour = {tour} />
                </div>
              )
            }
            )
          }
          </div>
```

```
        </div>
      </div>
    )
  }
}
// Вывод отрисованных шаблонов в отведенное для них место на стра-
нице index.php
ReactDOM.render(<App />, document.getElementById('root'))
```

Вы можете заметить, что при описании классов элементов HTML в React используется параметр className вместо class. При этом в CSS такие классы описываются также как обычно.

В результате после уточнения таблицы стилей, связанных с описанными классами элементов компонента, при запуске страницы *index.php* получим примерно такой результат:



За счет передачи параметра в URL гиперссылки мы получаем переход на соответствующую страницу tour.php:



Практическая работа № 9
СОЗДАНИЕ САЙТА В СРЕДЕ WORDPRESS

Цель: получить представлении о технологии создания сайтов с применением CMS систем.

Инструментарий: 1) любой современный браузер; 2) пакет разработчика Open Server.

Задания

1. Создать сайт, реализующий примерно те же функции, что и сайт, созданный в предыдущих практических работах с использованием возможностей CMS WordPress.
2. Создать собственный сайт с использованием той же технологии.

Выполнение задания 1

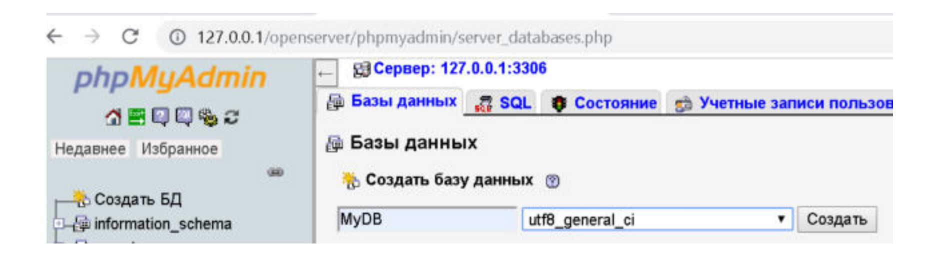
Шаг 1. Создание базы данных сайта.

WordPress — это CMS система, которая хранит все данные сайта в базе данных MySql. Поэтому нам необходимо создать пустую базу данных, которая будет подключена к будущему сайту

Запустим локальный сервер OpenServer, после его запуска выберем в меню *Дополнительно* пункт **PHPMyAdmin**. В браузере откроется программа для создания базы данных.

При входе система запросит логин и пароль, введем логин — root, а поле «Пароль» оставим пустым.

Базу данных называем, например, “myDB”, устанавливаем кодировку utf-8 general-c и нажимаем кнопку «Создать», как показано ниже:



Появится окно «MyDB была создана». При этом нужно помнить, что если не указывать ничего иного, то у этой БД имеется пользователь с максимальными правами под именем root, у которого по умолчанию нет пароля.

Шаг 2. Установка WordPress.

Скачиваем CMS-систему со страницы https://ru.wordpress.org/wordpress-5.0.8-ru_RU.zip. На момент написания пособия актуальной версией была 5.4. Распаковываем содержимое этого архива в папку, которая соответствует имени вашего сайта (например «new»): ..\OSPanel\domains\new.

Обратите внимание! Папки *wp-admin*, *wp-content*, *wp-includes* и другие файлы должны располагаться по следующему адресу:» ***..\OSPanel\domains\new***“, а **НЕ В ПАПКЕ *..\OSPanel\domains\wordpress\new.***

Указываем в адресе браузера путь: http://localhost/new/. Должно появиться окно начала настройки WordPress.

Выбираем русский язык

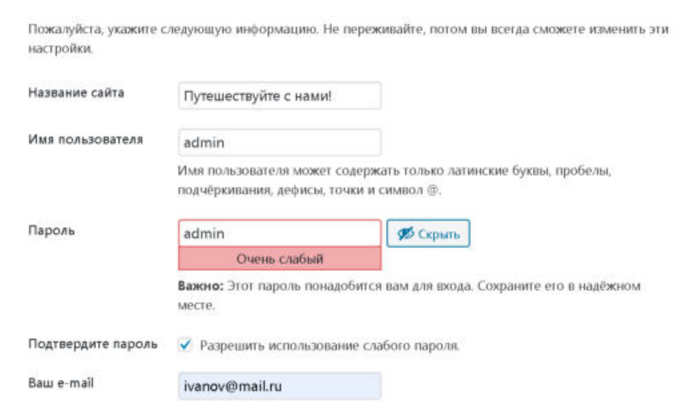
Указываем следующие настройки:

Below you should enter your database connection details. If you're not sure about these, contact your host.

Database Name	MyDB	The name of the database you want to use with WordPress.
Username	root	Your database username.
Password		Your database password.
Database Host	localhost	You should be able to get this info from your web host, if `localhost` doesn't work.
Table Prefix	wp_	If you want to run multiple WordPress installations in a single database, change this.

Submit

Далее, запускаем установку, снова подтверждаем выбор языка и заполняем следующее окно так:



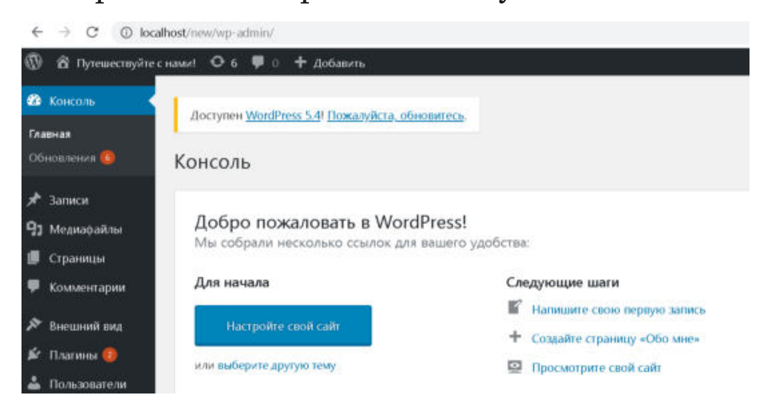
Пожалуйста, укажите следующую информацию. Не переживайте, потом вы всегда сможете изменить эти настройки.

Название сайта	Путешествуйте с нами!
Имя пользователя	admin Имя пользователя может содержать только латинские буквы, пробелы, подчёркивания, дефисы, точки и символ @.
Пароль	admin Скрыть Очень слабый **Важно:** Этот пароль понадобится вам для входа. Сохраните его в надёжном месте.
Подтвердите пароль	✓ Разрешить использование слабого пароля.
Ваш e-mail	ivanov@mail.ru

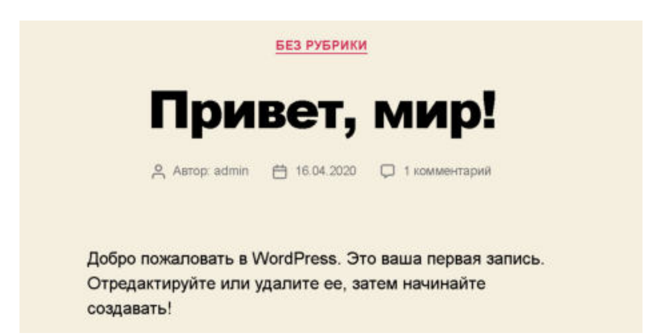
Пароль необходимо ввести и запомнить, он понадобиться для входа в административную панель.

Теперь мы можем работать в двух режимах — в режиме администрирования сайта и в режиме его просмотра.

Для перехода к администрированию требуется вводить в адресной строке: new/wp-admin. Результат:



Для просмотра сайта нужно ввести адрес new/. При этом результат будет таким:



Для перехода между этими режимами также могут использоваться две кнопки в верхнем меню программы.

Итак, сайт уже оформлен в соответствии с некоторой темой оформления, содержит одну запись «Привет, мир!» и имеет некоторые элементы управления.

Далее предлагается следующий план создания сайта:

1) выбрать и настроить тему оформления для сайта;
2) создать несколько статических страниц;
3) создать несколько записей, которые будут размещены на главной странице в виде новостей;
4) настроить меню сайта;
5) настроить дополнительные возможности при помощи виджетов;

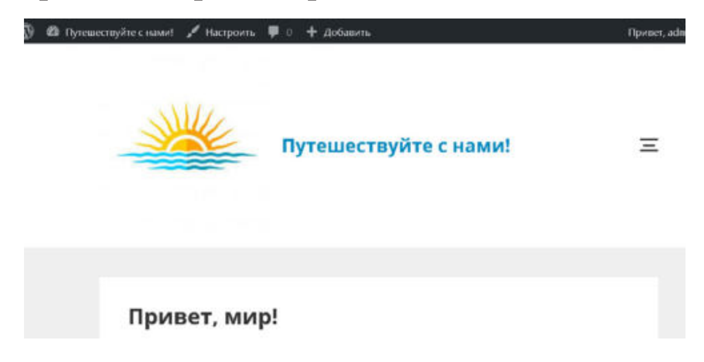
6) настроить дополнительные возможности с помощью плагинов.

Шаг 3. Настройка темы сайта

Если перейти в консоли по пунктам ***Внешний вид Темы*** вы можете увидеть встроенные темы. Если ни одна из предлагаемых тем для оформления сайта вам не подходит, можно по кнопке ***Добавить новую*** просмотреть и добавить одну из предлагаемых тем или закачать и установить новую тему. Бесплатные темы WordPress можно предварительно найти в Интернете, скачать и сохранить в виде zip-файла. В данном случае была применена одна из встроенных тем travel diaries.

Через меню ***Внешний вид*** → ***Темы*** → ***Настройки*** можно дополнительно задать варианты оформления и главной и остальных страниц сайта. Например, на главной странице отображается главная страница и область для дополнительных функций (сайдбар) справа, а на остальных — страница занимает всю горизонтальную область, а сайдбара нет. Эти настройки зависят от выбранной темы.

Теперь в режиме просмотра сайт выглядит так:

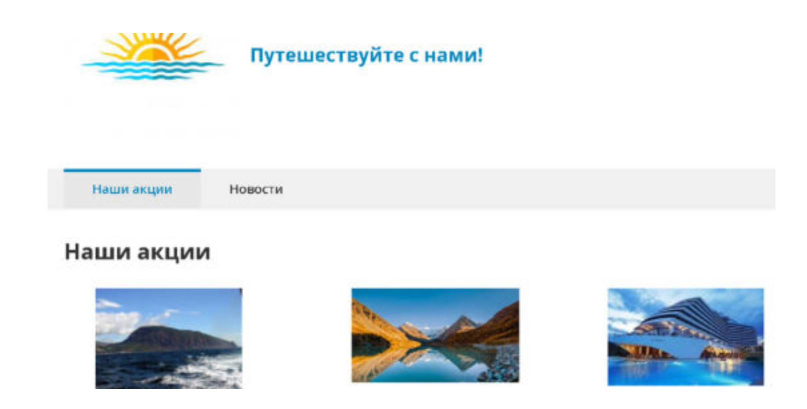


Шаг 4. Создание статических страниц.

Сайт состоит из статических и динамических страниц.

Для создания любой статической страницы необходимо выбрать в консоли ***Страницы*** → ***Добавить новую***. После этого можно указать название страницы, *которое в дальнейшем станет пунктом меню* для перехода на эту страницу, и создать любое содержание страницы, включая текст, рисунки, гипертекстовые ссылки и т. д.

Важно, что при добавлении элементов в текст страницы (кнопка «+») можно создать несколько колонок, которые нам нужны для поддержки предыдущего дизайна. Для сохранения страницы необходимо нажать на кнопку ***Опубликовать***.



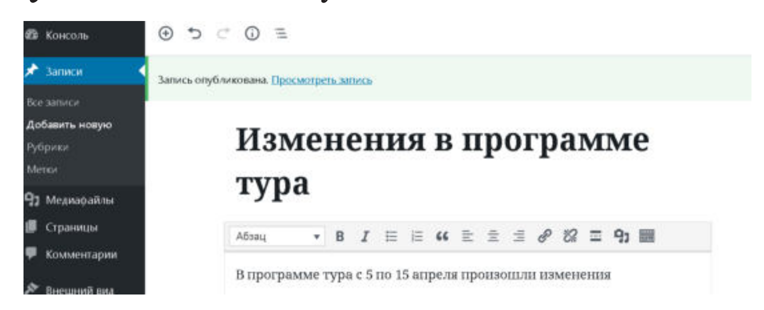
Таким же образом можно создать и другие страницы. Список всех страниц можно просмотреть, отредактировать настройки и содержимое каждой из них через меню ***Страницы Все страницы.***

Шаг 5. Создание записей.

Кроме статических страниц большинство современных сайтов содержат новостные страницы, которые обновляются автоматически, по мере добавления в базу данных новостей или записей.

Перед началом ввода записей нужно продумать, нужно ли вам, чтобы пользователи могли их комментировать, в каком виде они будут отображаться и т. д. Эти и другие настройки можно указать через пункты консоли ***Настройки*** → ***Написание/Чтение/Обсуждение***.

Для создания записей выбираем в консоли ***Записи*** → ***Добавить новую*** и заполняем нужные поля:



Таким образом, по мере добавления новостей последние будут отображаться на специальной странице новостей. Для этого нужно создать специальную страницу, в макете которой указать тип ***Блог.***

Шаг 6. Настройка меню сайта.

На сайте может быть одно или несколько меню, это зависит от выбранной темы. В данном случае мы настроим главное

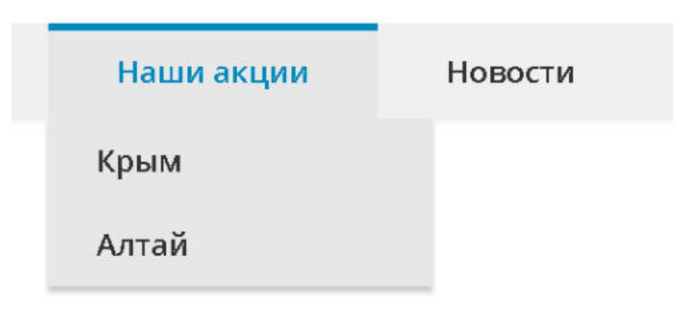
(верхнее меню) которое будет содержать ссылки на *все ранее созданные страницы* и динамическую страницу новостей.

Выберите ***Внешний вид*** → ***Меню.*** Введите название меню ***Главное*** и нажмите кнопку ***Создать меню.***

Выберите все страницы из списка слева и добавьте их в меню. Можно раскрыть любой из пунктов и изменить, например, текст ссылки.

Далее важно, что если вы хотите получить многоуровневое меню, то нужный пункт нужно просто сместить мышкой относительно того пункта, который является для него родительским:

Получим:

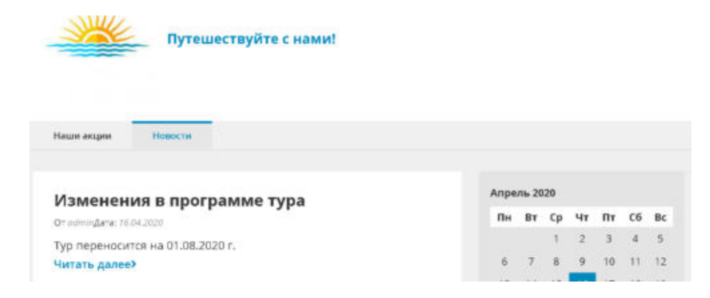


Шаг 7. Настройка виджетов.

Виджет — это небольшое приложение, созданное для реализации какой-нибудь полезной задачи. В WordPress под виджетами понимают встроенные подпрограммы, которые выполняют наиболее часто встречающиеся функции, такие как ведение архива новостей, календарь, поиск по сайту, регистрация пользователей и т. п. Особенность виджетов в WordPress в том, что их можно быстро разместить в любом месте страницы, которое предусмотрено используемой темой.

В WordPress есть встроенный набор виджетов. Например, если на странице новостей мы хотели бы справа показывать некоторую дополнительную информацию можно, через меню

Внешний вид → ***Виджеты*** перетащить, например, виджет «Календарь» в соответствующий сайдбар.



Вообще, все активные области, которые допускает эта тема, можно увидеть на странице виджетов.

Шаг 7. Авторизация и регистрация.

Для реализации функций регистрации и авторизации посетителей на сайте в WordPress имеются встроенные средства. Их настройка может быть выполнена через меню ***Настройки.***

После настройки возможностей оставлять отзывы пользователь может зарегистрироваться или авторизоваться и оставлять комментарии на странице записей.

Шаг 8. Настройка плагинов.

Плагин, в отличие от виджета, — это внешняя подпрограмма, которая выполняет дополнительные функции, не предусмотренные в базовом наборе WordPress. В стандартной поставке WordPress имеется несколько установленных плагинов, например Akismet Anti-Spam, который позволяет защитить ваш сайт от спама.

Для реализации дополнительных функций можно скачать плагины сторонних разработчиков.

Использованные источники

1. HTTP-запрос // CIT-Forum. — URL: http://citforum.ru/internet/cgi_tut/rqst.shtml.

2. *Вильямсон, Х.* Универсальный Dynamic HTML : Библиотека программиста / Х. Вильямсон. — Санкт-Петербург : Питер, 2001. — 304 с.

3. *Маркин, А. В.* Основы web-программирования на PHP / А. В. Маркин, С. С. Шкарин. — Москва : Диалог-МИФИ, 2012. — 252 с. — URL: http://biblioclub.ru/index.php?page=book&id=229742 (дата обращения: 12.09.2019).

4. *Фролов, А. В.* Базы данных в Интернете: практическое руководство по созданию Web-приложений с базами данных / А. В. Фролов, Г. В. Фролов. — Изд. 2, испр. — Москва : Издательско-торговый дом «Русская Редакция», 2000. — 448 с.

5. Общие сведения об Internet/Intranet // CIT-Forum. — URL: http://citforum.ru/internet/intranet_app/interintr_01.shtml.

Список рекомендованных источников

1. *Бейли, Л.* Изучаем PHP и MySQL / Л. Бейли, М. Моррисон. — Москва : Издательство Э, 2010. — 786 с.

2. Краткий справочник и учебник по JavaScript. — URL: http://www.wisdomweb.ru/JS/javascript-first.php (дата обращения: 10.01.2020).

3. Краткий справочник по Bootstrap. — URL: https://webref.ru/layout/bootstrap (дата обращения: 10.01.2020).

4. *МакГрат, М.* PHP7 для начинающих с пошаговыми инструкциями / М. МакГрат. — Москва : Издательство Э, 2018. — 256 с.

5. Справочник по CSS. — URL: https://webref.ru/css (дата обращения: 10.01.2020).

6. Справочник по языку HTML. — URL: https://webref.ru/html (дата обращения: 10.01.2020).

7. Справочник по языку PHP. — URL: https://www.php.net/manual/ru/langref.php (дата обращения: 10.01.2020).

8. Технология разработки интернет ресурсов: курс лекций / авт.-сост. И. А. Журавлёва. — Ставрополь : СКФУ, 2018. — 171 с. — URL: http://biblioclub.ru/index.php?page=book&id=562579 (дата обращения: 10.01.2020).

9. *Тузовский, А. Ф.* Проектирование и разработка web-приложений : учебное пособие для академического бакалавриата / А. Ф. Тузовский. — Москва : Издательство Юрайт, 2019. — 218 с. — URL: https://urait.ru/bcode/433825 (дата обращения: 10.01.2020).

10. *Фримен, Э.* Изучаем программирование на JavaScript / Э. Фримен, Э. Робсон. — Санкт-Петербург : Питер, 2017. — 637 с.